Comprando esta edição tenha acesso na WEB a:
+330 vídeo-aulas +150 artigos e 6 cursos online

www.clubedelphi.net Feita para desenvolvedores

ClubeDelphi

Ano 7 · Edição 83 · R$9,90 DevMedia

Mini-curso

Construa uma aplicação Web usando ASP.NET, UML e ECO

ISSN 1517990-7

- Coluna: Ask the Expert!

Na WEB:

Aulas sobre POO

Confira no portal ClubeDelphi PLUS **7 vídeo-aulas** sobre POO, UML e ECO com Delphi!

Firebird
Conheça a nova ferramenta nbackup do FB 2.0 e saiba como fazer backups incrementais

Boa Idéia
Saiba como resolver problemas de corrupção no banco de dados

Mão na Massa
Saiba como trabalhar com scanner em sua aplicação Delphi, fazendo a captura de documentos e gravação no BD

Firebird
Tudo o que você sempre quis saber sobre Generators

Delphi e ASP.NET
Trabalhando com imagens em aplicações Web

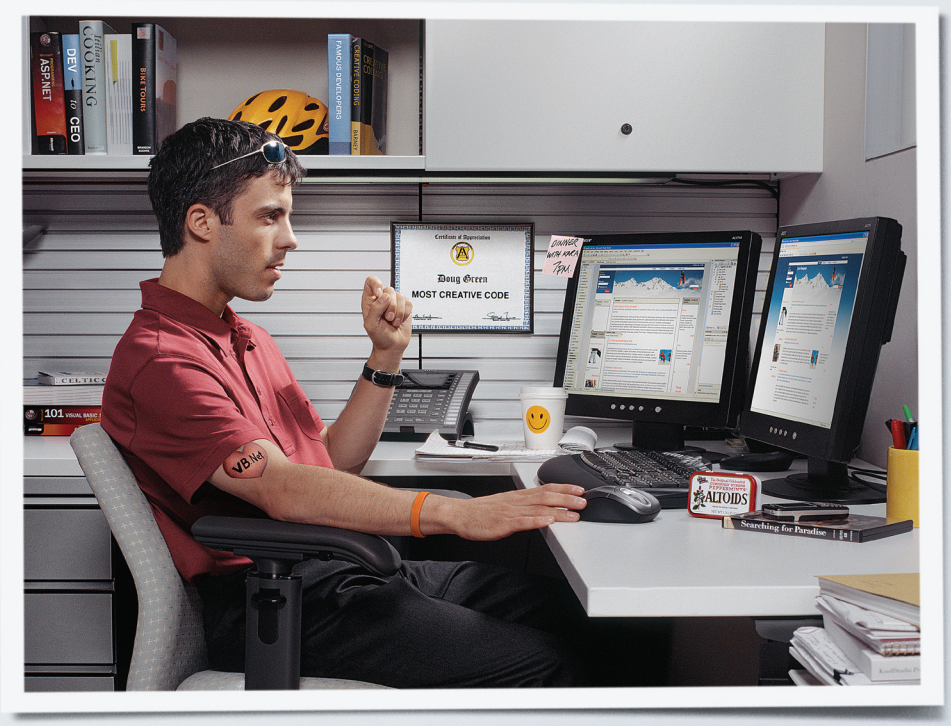
Delphi e Outlook
Mala direta com e-mails personalizados - Parte 2

POO
Modelagem UML com Delphi

Seu potencial. Nossa inspiração.™

Microsoft

McCANN



Novo Visual Studio® 2005. A diferença é óbvia.

Descobriu a diferença entre as duas imagens acima? Quando começar a desenvolver, você verá. O novo Visual Studio® 2005 possui mais de 400 novos recursos que o ajudarão a construir aplicações Web e acelerar seu projeto em plataforma .Net, reduzindo o tempo perdido em tarefas entediantes e repetitivas. Assim, você poderá trabalhar na criação de códigos de qualidade. Saiba mais sobre o novo Visual Studio® 2005 acessando **www.msdn.com.br/visualstudio2005**

Microsoft® Visual Studio® 2005

© 2006 Microsoft Corporation. Todos os direitos reservados. Microsoft, Visual Studio, o logo do Visual Studio e "Seu potencial. Nossa inspiração" são marcas registradas ou marcas da Microsoft Corporation nos Estados Unidos e/ou em outros países.

NÃO PERCA

PORTAL DO ASSINANTE

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:

- Mais de 330 Vídeo Aulas!
- 6 cursos online!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 140 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse www.devmedia.com.br/clubedelphi/portal.asp e utilize as informações abaixo:

Login: DVM.CD

Senha: A54BR

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Corpo Editorial

Diretor Editorial e Administrativo
Gladstone Matos
gladstone@neoficio.com.br

Designer
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Antonio Xavier
jaxs_design@yahoo.com.br

Articulistas desta edição
Paulo Roberto Quicoli, Fernando Sarturi Prass, Fabrício Desbessel, Frank Ingermann, Paul Vinkenoog, Adriano Santos, Murilo Costa Monteiro Filho, Alberto Germano Sevarolli Trevezani e Marco Antônio Pereira Araújo

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Luciano Pimenta
lucianopimenta@clubedelphi.net

Revisão
Fabio Correa
fabiocorrea@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:
Kaline Dolabella
publicidade@devmedia.com.br

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Thiago Andrade – Atendimento ao Leitor
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Parcerias

Para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:
Jeff Wendell
jeff@devmedia.com.br

ASSESPRO PARANÁ

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão! Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

Luciano Pimenta - Editor do Site
lucianopimenta@clubedelphi.net

ClubeDelphi

Ano 7 - 83ª Edição - 2007 - ISSN 1517990-7 **Impresso no Brasil**

Editorial

O ECO é um framework que permite a construção de aplicações usando o modelo MDA (Model Driven Architecture). Além disso, é um framework de persistência que oferece um mecanismo simples e elegante para mapeamento O/R (objeto relacional). Nesta edição, o Paulo Quicoli inicia um mini-curso totalmente prático que vai mostrar como usar o ECO III em uma aplicação real.

Ainda nesta edição, na coluna "Boa Idéia", o Fernando Prass reporta uma boa idéia que teve ao precisar recuperar um banco de dados Firebird corrompido.

Sobre desenvolvimento Web, o Fabrício Desbessel mostra como trabalhar com imagens em aplicações ASP.NET. Aprenda a responder a requisições HTTP enviando imagens ao browser, redimensionando-as e adicionando uma marca d'água.

Um Generator é um recurso que o banco de dados Firebird utiliza para permitir a criação de campos estilo auto-incremento (lembra do Paradox?). Existem vários detalhes envolvidos na utilização deste recurso, como destaca Frank Ingermann em seu excelente e completo artigo sobre Generators.

O Paul apresenta a nova ferramenta nbackup do Firebird 2.0, destinada a criação de backups, inclusive incrementais. Saiba como utilizar o nbackup com seus vários parâmetros, para fazer o backup do seu banco de dados.

O Adriano Santos coloca a mão na massa e mostra como criar um utilitário para gerenciamento eletrônico de documentos em Delphi, que permite interagir com um scanner, capturando documentos e gravando em um banco de dados Firebird.

O Murilo continua a série de artigos que mostra como enviar e-mails através do Delphi e Outlook. Neste mês apresenta como construir um template processor HTML para envio de mala direta personalizada.

Para finalizar, o Marco Antônio e o Alberto Germano mostram como utilizar a integração com o Together no Delphi 2006, para a modelagem UML de sistemas. Aprenda a criar diagramas de caso de uso com base em requisitos, diagramas de classes e de sequência.

Bom leitura e sucesso com o Delphi.

Guinther Pauli – guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

ÍNDICE

06 - Desenvolvendo uma Loja Virtual com ASP.NET, UML e ECO – Parte 1
Paulo Roberto Quicoli

12 - Copiando registros de um banco de dados corrompido
Fernando Sarturi Prass

18 - Como trabalhar com imagens em aplicações ASP.NET
Fabrício Desbessel

22 - Um guia de como e quando utilizar Generators no Firebird
Frank Ingermann

30 - Ferramenta nbackup do Firebird 2.0
Paul Vinkenoog

36 - Como escanear e armazenar documentos com Delphi
Adriano Santos

40 - Envio automatizado de emails com Delphi e Outlook - Parte 2
Murilo Costa Monteiro Filho

44 - Modelagem UML com Together
Alberto Germano Sevarolli Trevezani e Marco Antônio Pereira Araújo

Ask The Expert

Perguntas e Respostas

Button com Caption de múltiplas linhas

Gostaria de "quebrar" o *Caption* do meu botão. Tem como fazer?

Estevan Cechela

A **Listagem 1** configura os *Captions* de todos os *Buttons* de determinado *Parent* para permitirem múltiplas linhas.

Listagem 1. Quebrando o Caption dos Buttons

```
procedure SetMultiLineButton(AParent: TWinControl);

var
  i: integer;
  ah: THandle;

begin
  for i := 0 to AParent.ControlCount - 1 do

  if (AParent.Controls[i] is TButton) then
  begin
    ah := (AParent.Controls[i] as TButton).Handle;
    SetWindowLong(ah, GWL_STYLE, GetWindowLong(
      ah, GWL_STYLE) OR BS_MULTILINE) ;
  end;

end;
```

No evento *OnCreate* do formulário, por exemplo, você pode usar o seguinte código:

```
SetMultiLineButton(Form1);
```

Veja na **Figura 1** o exemplo.

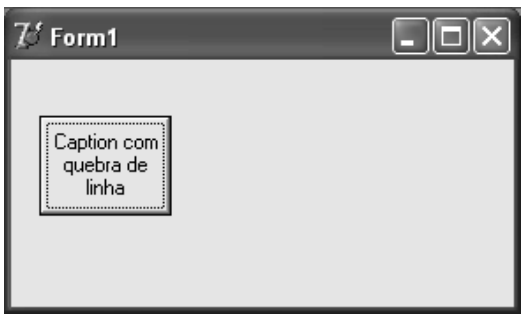


Figura 1. Caption do Button com quebra de linha

Como alinhar itens de menu à direita

Olá, primeiramente, parabéns pelo trabalho que vocês estão fazendo para o bem da comunidade Delphi. Dias desses, estava trabalhando em um cliente, e vi um software onde o menu Help do mesmo, ficava alinhado a direita da tela. Gostaria de saber se posso fazer isso no Delphi.

Antônio José Silva

Olá Antônio, obrigado. Para fazer um teste, primeiramente adicione um *MainMenu* ao formulário, com alguns itens. Nomeie um item de menu como "HelpMenuItem". Use o código da **Listagem 2** no evento *OnCreate* do formulário.

Listagem 2. Alinhando o menu a direita do formulário

```
var
  mii: TMenuItemInfo;
  MainMenu: hMenu;
  Buffer: array[0..79] of Char;
begin
  MainMenu := Self.Menu.Handle;
  { Pega as propriedades do item de menu Help }
  mii.cbSize := SizeOf(mii);
  mii.fMask := MIIM_TYPE;
  mii.dwTypeData := Buffer;
  mii.cch := SizeOf(Buffer);
  GetMenuItemInfo(MainMenu, HelpMenuItem.Command, False, mii);
  { Configura as propriedades do item de menu Help }
  mii.fType := mii.fType or MFT_RIGHTJUSTIFY;
  SetMenuItemInfo(MainMenu, HelpMenuItem.Command, False, mii);
end;
```

Execute o projeto e veja que o item *Help* está alinhado à direita na barra de menu (**Figura 2**).

Figura 2. Alinhando o menu de Help a direta do formulário

Assinatura
ClubeDelphi PLUS
Mais conteúdo .NET por menos!

Novidades do ClubeDelphi PLUS

+330 vídeo aulas!
6 cursos online

www.clubedelphi.net/portal

Caro Leitor,

O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 330 vídeo aulas; ii) 6 cursos online; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 150 artigos exclusivos (que não foram publicados na revista)!;

Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).

Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.

Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!

Equipe DevMedia

.:: DESTAQUE ::.

Confira no portal ClubeDelphi PLUS 7 vídeo aulas sobre POO, UML e ECO com Delphi!:

Utilizando ECO em aplicações ASP.NET
http://www.devmedia.com.br/articles/viewcomp.asp?comp=1999

Usando Propriedades na Orientação a Objetos
http://www.devmedia.com.br/articles/viewcomp.asp?comp=4386

Together no Delphi 2006
http://www.devmedia.com.br/articles/viewcomp.asp?comp=2633

Herança visual no .NET Framework
http://www.devmedia.com.br/articles/viewcomp.asp?comp=2314

Herança Visual
http://www.devmedia.com.br/articles/viewcomp.asp?comp=4384

Introdução ao ECO com Windows Forms
http://www.devmedia.com.br/articles/viewcomp.asp?comp=556

Usando o ECO com ASP.NET
http://www.devmedia.com.br/articles/viewcomp.asp?comp=557

Últimas Vídeo-Aulas (Somente para assinantes)

Relatórios no Quick Report – Parte I a V
Veja nessas aulas de Fabricio Desbessel, várias dicas sobre relatórios no Quick Report.

Aplicações Windows Forms no Delphi 2006 - Parte V
Veja nessas aulas de Paulo Quicoli, como trabalhar com aplicações Windows Forms com Delphi 2006.

Gerenciando configurações de sistema - Parte I e II
Veja nessas vídeo aulas de Paulo quicoli, como gerenciar configurações de sistema.

Criando uma interface VCL multi-tab estilo IE7
Veja nessa vídeo aula de Guinther Pauli, como criar uma interface VCL multi-tab, estilo IE7.

Convertendo aplicações C# para Delphi for .NET
Veja nessa vídeo aula de Guinther Pauli, como converter aplicações C# em Delphi for .NET.

Transformações XML com a classe XslTransform
Veja nessa vídeo aula de Guinther Pauli, como realizar transformações XML com o XslTransform do .NET.

Sistema completo com Delphi 7, dbExpress e Firebird 2.0
Acompanhe a criação de um sistema completo de gerenciamento de locadora, usando Delphi, dbExpress e Firebird 2.0. Luciano Pimenta, mostrará todos os passos necessários para a criação do projeto em todas as fases de implementação, desde a criação do banco de dados, até a migração para .NET.

Desenvolvendo uma Loja Virtual com ASP.NET, UML e ECO – Parte 1

PAULO ROBERTO QUICOLI
(pauloquicoli@gmail.com)
é analista e programador da Control-M Informática. Trabalha com Delphi desde sua primeira versão, e Firebird, desenvolvendo aplicações cliente-servidor. Formado em Tecnologia de Processamento de dados pela FATEC, na cidade de Taquaritinga/SP

Nesta série de artigos, vamos utilizar o ECO III no desenvolvimento de um pequeno site de compras. Vamos explorar a composição do ambiente ECO, seus recursos, funcionalidades e facilidades oferecidas. Tudo isso de forma prática para que seja despertado em você leitor, o interesse pelo assunto.

Durante o desenvolvimento do site, não entrarei em detalhes sobre recursos que serão utilizados como, por exemplo, *User Controls*, isso porque quero manter o foco no ECO III. Contudo citarei artigos de outras edições da ClubeDelphi, bem como vídeo aulas, que explicam de maneira exemplar a criação dos mesmos.

Um pouco de história

Antes da existência do ECO tínhamos um framework chamado *Bold*, originalmente desenvolvido por uma empresa chamada BoldSoft. Em 1997 foi lançada a primeira versão do Bold para Delphi, na época, Delphi 2.

Com o surgimento do .NET, o ECO foi lançado e teve como base o Bold, podendo ser definido como uma evolução do Bold para o mundo .NET. Veja que eu disse *evolução* e não que foi um framework *portado*. Muitos dos conceitos que se aplicam ao ECO também se aplicam ao Bold.

O que é o ECO

O ECO faz parte de um conjunto de ferramentas destinadas à *Arquitetura Dirigida ao Modelo*, ou simplesmente MDA (Model Driven Architecture). A MDA possui um objetivo muito claro que é tornar independente a regra de negócio de uma aplicação.

Esses objetivos, especificações, padronizações e diretrizes são definidas pela OMG (*Object Management Group*), grupo responsável pela normatização de ferramentas e tecnologias relacionadas a objetos. Possuindo a UML como "linguagem" para representar um modelo de negócios Orientado a Objetos, permi-

te que esse mesmo modelo seja utilizado em aplicações Windows Forms, ASP.NET e até mesmo por Web Services, atingindo assim a independência do modelo sugerida pela MDA.

Contudo, existem alguns pontos que a UML não consegue cobrir, por exemplo, restrições que representam atributos de uma classe. Para suprir isso, foi desenvolvida a OCL (*Object Constraint Language*) e ela é largamente utilizada pelo ECO.

Um exemplo de utilização seria ter uma classe *TEleitor* e nela lançaríamos apenas pessoas que podem ser eleitores, portanto é lógico aplicarmos uma regra nessa classe onde apenas pessoas com 16 anos ou mais são possíveis eleitores. Para aplicar isso a uma classe utilizamos uma sentença OCL.

De modo geral podemos dizer que o desenvolvimento MDA e suas ferramentas enfatizam o modelo de negócios de uma aplicação. Na **Figura 1,** temos uma visão geral da composição de uma aplicação ECO.

Sobre os componentes, temos a definição:

• *ECOSpace*: Em tempo de execução atua como uma cache inteligente em memória, ficando responsável por carregar os objetos quando necessário e também por liberá-los da memória. O *ECOSpace* sempre está ligado, de forma automática ao *ECOPackage UML*, que contém o modelo de classes em questão. Além de gerenciar os objetos em execução, o *ECOSpace* expõe serviços como UNDO, OCL , transações etc.;

• *Persistence Mapper*: É responsável pela transformação de objetos em dados que serão salvos em algum meio persistente, como por exemplo um banco relacional, e pela recuperação desses mesmos dados na forma de objetos;

• *RootHandle*: Liga o *ECOSpace* à GUI em uso, que pode ser um Windows Form ou Web Form;

• *ExpressionHandle*: Através dele podemos acessar as classes do modelo UML, seja para exibição em um *DataGrid* ou para edição em *TextBox*. Para realização de *Data Binding* é utilizado como um *DataSource,* oferecendo ainda uma propriedade chamada *Expression,* onde especificamos que classe de modelo vamos utilizar. É também nessa propriedade que definimos as sentenças OCL;

• *CurrencyManagerHandle*: Funciona como um ponteiro indicando que objeto está em foco dentre um lista.

Metodologia

A aplicação desenvolvida neste mini-curso é desenvolvida sobre um modelo multicamadas, onde temos: banco de dados, aplicação web, camada de persistência e interface (browser). As vantagens dessa abordagem são muitas: separação lógica entre as partes da aplicação, facilidade de manutenção, reutilização de código, melhor uso de técnicas de POO, torna a lógica de negócio independente da arquitetura utilizada e tecnologias empregadas (ex.: ASP.NET, Windows Forms, Web Services), menor impacto na mudança de requisito e redução de custos.

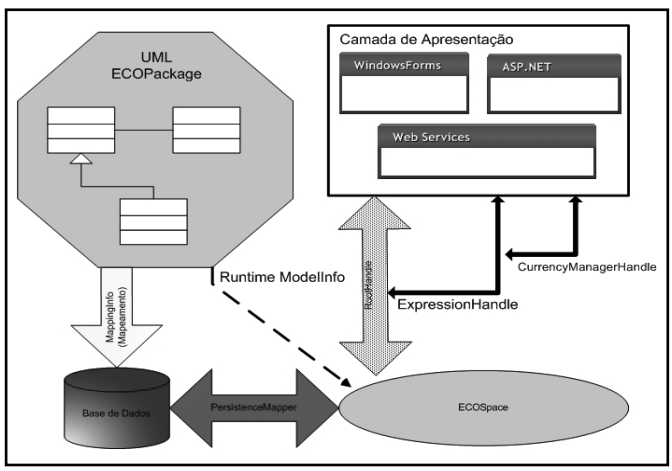


Figura 1. Visão geral de uma aplicação ECO

Iniciando o projeto

No BDS, crie um novo projeto através do menu *File>New>Other*, item *Delphi for .NET Projects>ECO ASP.NET Web Application*. Altere o nome do projeto para "ECOStore", escolha o servidor Web e clique em OK.

Vamos renomear o *WebForm1* para 'Home.aspx", para isso selecione o arquivo no *Project Manager*, clique com o botão direito e escolha *Rename*. Na **Tabela 1** temos a explicação de alguns arquivos que compõem o projeto.

Arquivo	Característica
ECOStore.bdsproj	Arquivo de projeto.
Package_1Unit.pas	Contém as classes do modelo (modelo de domínio).
ECOStoreEcoSpace.pas	Implementa a classe EcoSpace.
Home.aspx	Arquivo de design de um Web Form.
Home.pas	Arquivo que contém o code-behind, a lógica do Web Form.
Global.asax	Implementa funções globais.
Web.config	Arquivo de configuração.

Tabela 1. Composição de um projeto ECO

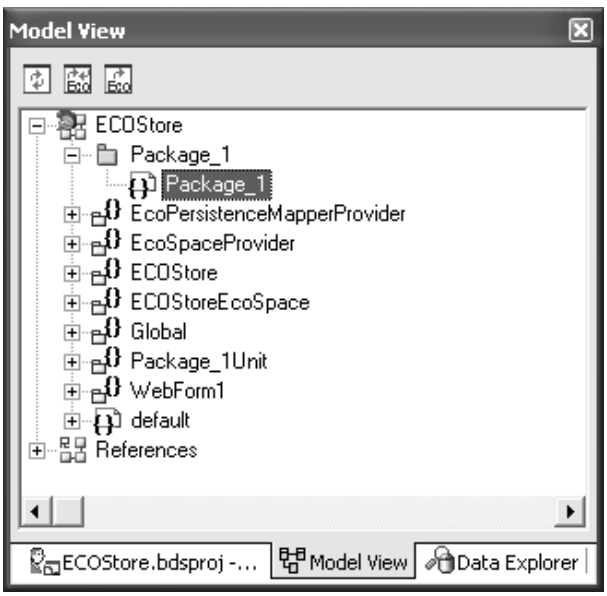


Figura 2. Model View do projeto

Modelando as classes

No *Project Manager* ative a guia *Model View* e dê um duplo clique sobre *Package_1* (**Figura 2**). O designer de classes é então apresentado.

A partir da *Tool Palette*, arraste um *ECO-Class* e altere seu nome para "Usuario". Para adicionar as propriedades (atributos) da classe utilizamos o menu contextual (clique com o botão direito sobre a classe). Adicione as propriedades conforme a **Figura 3**.

Adicione mais duas *ECOClass* e altere seus nomes respectivamente para "Produto" e "Categoria", adicionando posteriormente os atributos, conforme visto na **Figura 4**.

Figura 3. Classe Usuario

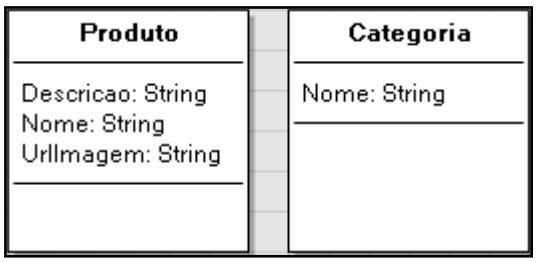


Figura 4. Classes Produto e Categoria

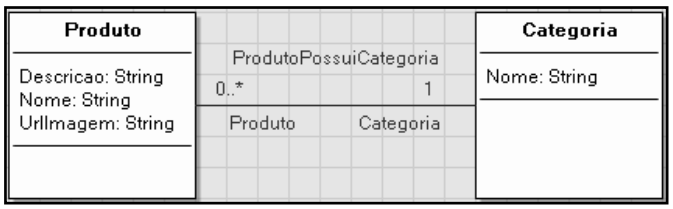


Figura 5. Configuração da associação das classes

Vamos agora adicionar uma associação entre as classes. Na *Tool Palette* clique sobre o item *Association*. Clique sobre a classe *Produto*, mantenha o botão do mouse pressionado e arraste até a classe *Categoria*, soltando então o botão.

> **Nota:** *A direção que utilizamos para arrastar a associação é muito importante.*

Devemos agora configurar a associação, para isso, clique na linha da associação e pressione F11 para chamar o *Object Inspector*. Nele vamos editar a propriedade *Name* para "ProdutoPossuiCategoria". Em *End1* e *End2* vamos alterar as propriedades *Name* para "Produto" e "Categoria", respectivamente. Em *Multiciplity*, escolha "0..*" e "1" (**Figura 5)**.

Com isso dizemos que um produto deve possuir uma categoria e que uma categoria pode constar em nenhum ou vários produtos. Criaremos agora uma classe base que conterá a propriedade ID, que será comum a todas as outras classes.

Adicione uma *ECOClass* ao diagrama e chame-a de "ClasseBase". Adicione a ela uma propriedade de nome "Id" de tipo *AutoInc*. Na *Tool Palette* clique no item *Generalization/Implementation*. Observe que o ponteiro do mouse muda. Clique sobre a classe *Usuario* e arraste até *ClasseBase*. Repita o processo para as classes *Produto* e *Categoria*. O resultado final deve ser semelhante ao da **Figura 6**.

Gerando o banco de dados

Baseando-se no modelo UML, o ECO pode criar o banco de dados. Acessando o *Project Manager* abra o item *TEcoPersistenceMapperProvider* e ative a guia *design*. Para persistir os dados vou utilizar o SQL Server 2005 Express, onde já criei um banco de dados chamado "EcoStore" (apenas o BD vazio, pois as tabelas serão criadas pelo ECO).

> **Nota:** *Para saber como instalar o SQL Server 2005 Express veja na edição 74 um artigo de Everson Volaco.*

Adicione um *PersistenceMapperSqlServer* que está contido em *Enterprise Core Objects* da *Tool Palette*. Adicione um *SqlConnection*, da categoria *Data Components*, configurando a conexão com o banco de dados. Adicione também um *SyncHandler* da categoria

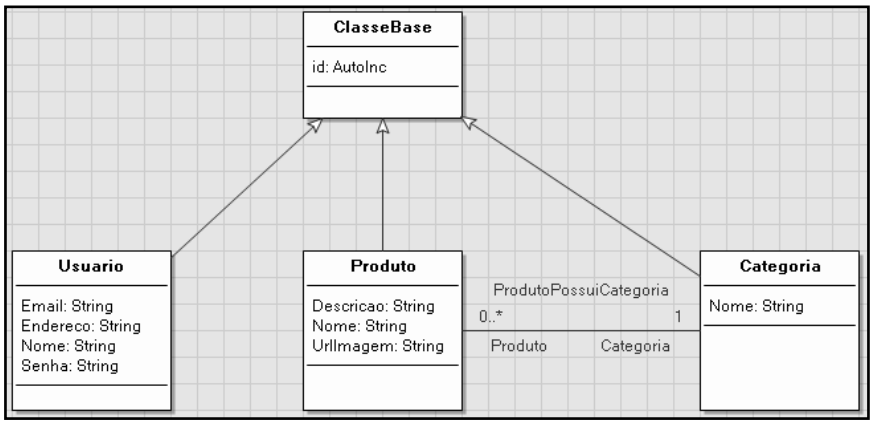


Figura 6. Digrama de classes inicial

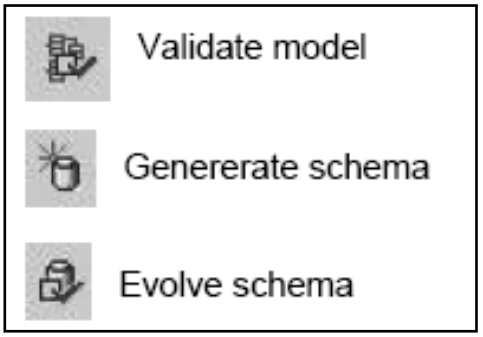


Figura 7. Botões para geração de banco de dados

Enterprise Core Objects. Selecione o *PersistenceMapperSqlServer* e ajuste a propriedade *SyncHandler* para o objeto que adicionamos.

Clique na área vazia (em branco) do *EcoPersistenceMapperProvider* e pressione F11 para trazer o *Object Inspector*. Vamos ligá-lo ao *ECOSpace* através da propriedade *EcoSpaceType*.

Nota: *Neste ponto será necessário dar um* ***Build*** *na aplicação.*

Abra o *TECOStoreEcoSpace* e clique com o botão direito do mouse sobre alguma área em branco. No menu de contexto escolha *Select packages*, escolha o *Package 1* e coloque-o no item *Selected UML Packages*. Por fim, compile a aplicação. Com isso, informamos ao ECO que vamos utilizar as classes que estão definidas nesse *package*.

Observe que na parte inferior da janela *EcoPersistenceMapperProvider* temos alguns botões que serão utilizados para a geração do banco de dados, vamos usar três deles (**Figura 7**).

Suas características são:

• *Validate Model*: Realiza uma verificação no modelo existente, identificando erros em expressões OCL, parâmetros ilegais etc. Será muito utilizado na aplicação;

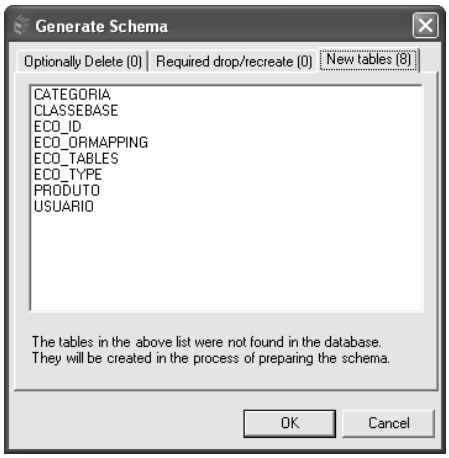


Figura 8. Janela Generate Schema

• *Generate Schema*: As tabelas serão criadas e todas as tabelas existentes serão excluídas, portanto utilize essa opção com muita atenção para que não se perca informação alguma. Antes de serem criadas será exibida uma janela mostrando as operações que serão feitas no banco de dados;

• *Evolve Schema*: O banco de dados será atualizado para representar as mudanças feitas no modelo UML. Antes de executar um *Evolve* faça uma cópia de segurança do banco de dados.

Clique em *Validate Model*. Após a validação uma "dica" na janela *Messages* será exibida. No momento podemos ignorá-la porque não afetará nossa aplicação. Depois de validado podemos gerar o banco.

Clique em *Generate Schema* e uma janela será exibida, mostrando o que será feito no banco de dados (**Figura 8**). Clique em OK e o esquema será criado no banco. Observe na janela *Messages* a existência de um log do que foi feito.

ClubeDelphi PLUS

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como trabalhar com o SQL Server Express no Delphi.

www.devmedia.com.br/articles/viewcomp.asp?comp=2448

ClubeDelphi PLUS

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como trabalhar com User Controls no Delphi for .NET.

www.devmedia.com.br/articles/viewcomp.asp?comp=2220

Figura 9. User Control Cabecalho.ascx

Criando a interface

Com as classes iniciais e banco prontos, podemos partir para a criação das interfaces, os Web Forms. Vamos criar um *User Control* (dando o nome de "cabecalho.ascx") que será o cabeçalho das páginas. Nele constará o menu de navegação da página que é composto por cinco *Hyperlinks*, conforme visto na **Figura 9**.

> **Nota:** *Utilizar* ***Users Controls*** *pode melhorar o desempenho da aplicação, porque o ASP.NET pode fazer cache desse tipo de controle.*

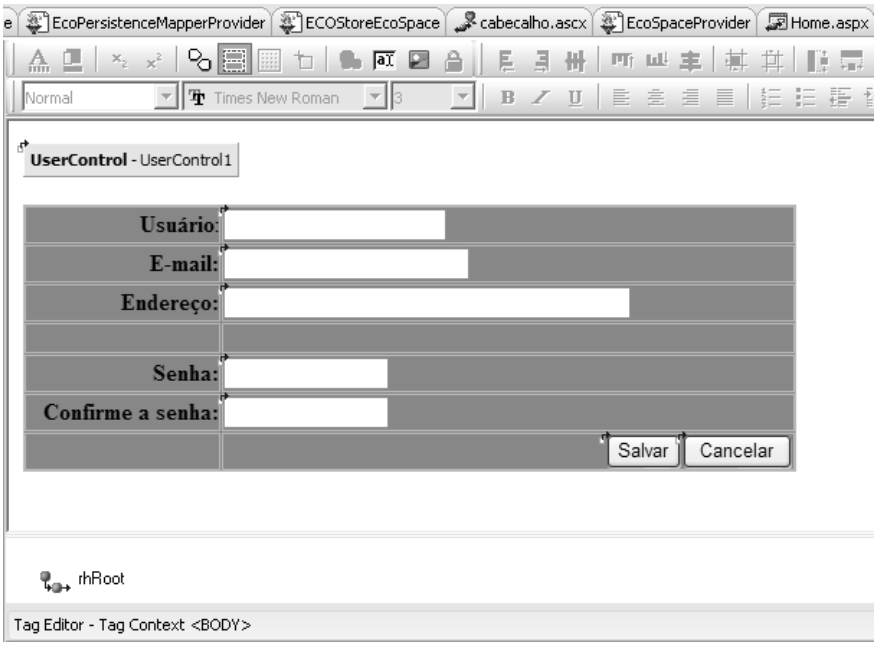


Figura 10. Cadastro de usuários

Figura 11. Cadastrando um usuário usando ECO

No *User Control* ajuste a propriedade *NavigateUrl* do primeiro *Hyperlink* para "Home.aspx". Adicione o *User Control* na página principal da aplicação.

Cadastro de usuários

Como todo site de vendas, só pode comprar quem estiver cadastrado no site e essa verificação é realizada no encerramento da compra. Nossa aplicação não será diferente. Vamos criar um novo ECO WebForm acessando o menu *File>New>Other>New ASP.NET Files>ECO ASP.NET Page*, renomeie como "CadUsuario.aspx".

Veja que automaticamente é adicionado ao Web Form um *RootHandle*. Devemos ligá-lo ao *ECOSpace* através da propriedade *EcoSpaceType*. Nesse momento adicione nosso *User Control* e modele a página como na **Figura 10.**

Nos *TextBoxes* que se referem à senha e confirmação, devemos mudar a propriedade *TextMode* para *Password*, dessa forma o que for digitado no controle terá uma máscara ("*").

Adicione um *RequiredFieldValidator*, posicionando-o ao lado no campo do nome do usuário. Altere sua propriedade *ControlToValidate* para *txtUsuario* e em *ErrorMessage* digite: "O nome do usuário é obrigatório".

Vamos validar agora o e-mail digitado através de um *RegularExpressionValidator*. Adicione-o ao lado do *TextBox* do e-mail, mudando a propriedade *ControlToValidade* para *txtEmail*, *ErrorMessage* para "E-mail inválido" e em *ValidationExpression* escolha a expressão que se refere à validação de e-mails (*Internet E-mail Address*).

É preciso validar também se o usuário digitou a senha de forma correta no campo *Confirme a senha*, para isso, vamos adicionar um *CompareValidator*, posicionando-o ao lado do *txtConfirmaSenha*. Altere as propriedades *ControlToCompare* para *txtSenha*, *ControlToValidate* para *txtConfirmaSenha* e em *ErrorMessage* digite "Senhas não conferem".

Com nosso cadastro validado, vamos codificar os botões *Salvar* e *Cancelar*. Primeiramente adicione ao *CadUsuario.aspx* uma referência ao *Package_1* (inclua no *uses* a unit *Package_1Unit*), que contém nossas classes e posteriormente adicione o código da **Listagem 1** ao *Click* do *Salvar*.

Você pode estar se perguntando de onde tirei a variável *EcoSpace* e o *UpdateDatabase*. Quando criamos um *ECO ASP.NET Page*, o assistente do Delphi já cria algumas rotinas para facilitar nossa vida. Veja na **Listagem 2** a implementação do *UpdateDatabase*.

O objeto *EcoSpace* representa uma instância do *EcoSpace* em uso e esse oferece alguns

métodos como o *UpdateDatabase*, que atualiza no banco de dados todos os objetos que foram alterados, criados ou excluídos. Veja na **Figura 11** a aplicação em execução, criando um usuário.

Para finalizar, no *Cancelar* altere a propriedade *CausesValidation* para *False* e adicione no evento *Click* o seguinte código:

```
Response.Redirect('Home.aspx');
```

Não vamos esquecer de no *User Control*, no *Hyperlink* referente ao cadastro de usuários, alterar a propriedade *NavigateUrl* para "CadUsuario.aspx". Execute a aplicação e verá que tudo funciona perfeitamente.

Conclusão

Após uma introdução ao ECO, vimos que ele realmente nos livra das tarefas mais "chatas" como lidar com SQL, operações de inclusão e exclusão, mapeamento O/R, visto que todo esse trâmite é gerenciado por ele.

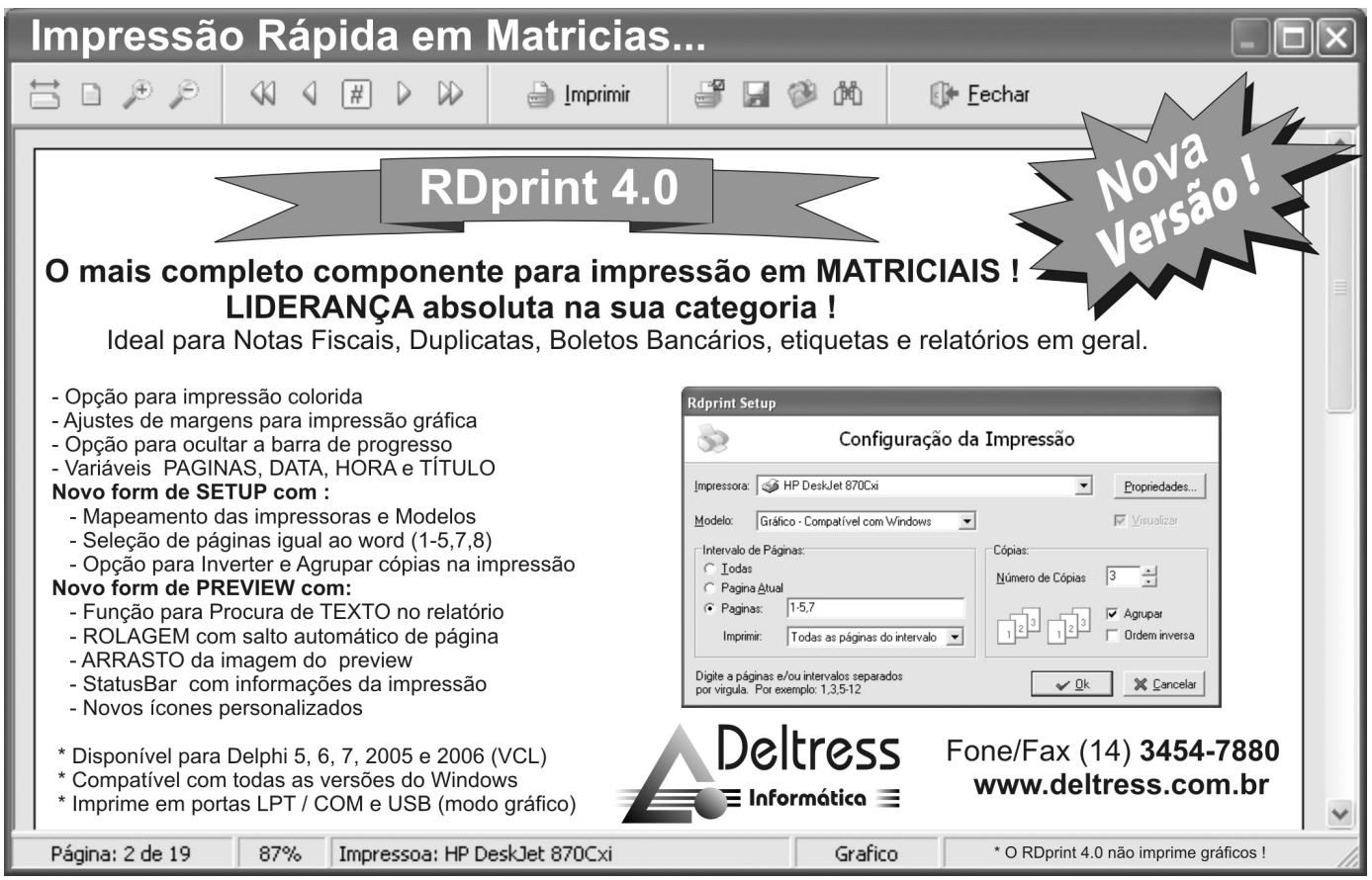
Realmente o ECO cumpre um dos objetivos da MDA que é permitir ao desenvolvedor manter o foco no modelo de negócios. Nos próximos artigos vamos continuar a explorar mais recursos do ECO como a OCL. Veremos também como transferir objetos entre páginas, portanto, não perca a próxima edição. Abraço! ■

Listagem 1. Salvando o usuário

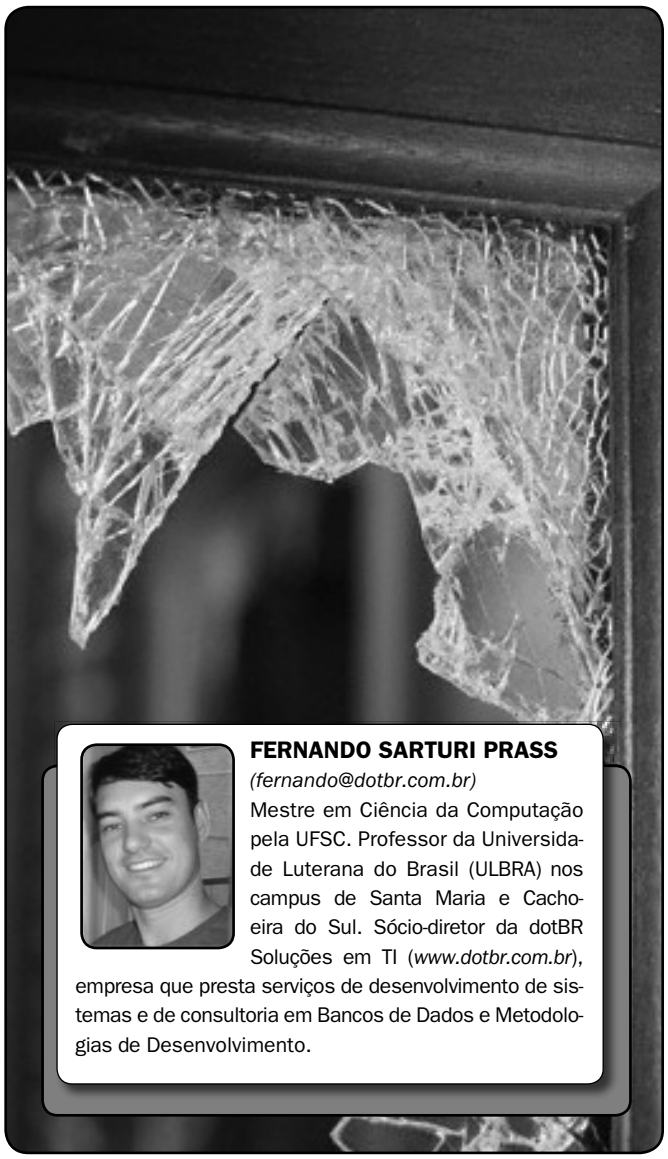
```
procedure TWebForm1.btnSalvar_Click(
  sender: System.Object; e: System.EventArgs);

var
 obj: Usuario;

begin
  obj := Usuario.Create(EcoSpace);
  obj.Nome := txtUsuario.Text;
  obj.Endereco := txtEndereco.Text;
  obj.Email := txtEmail.Text;
  obj.Senha := txtSenha.Text;
  UpdateDatabase;
  Response.Redirect('Home.aspx');
end;
```

Listagem 2. Método UpdateDatabase

```
procedure TWebForm1.UpdateDatabase;

var
  PersistenceService: IPersistenceService;
  DirtyListService: IDirtyListService;

begin
  PersistenceService := EcoSpace.GetEcoService(
    typeof(IPersistenceService)) as
    IPersistenceService;
  DirtyListService := EcoSpace.GetEcoService(
    typeof(IDirtyListService)) as IDirtyListService;
  PersistenceService.UpdateDatabaseWithList(
    DirtyListService.AllDirtyObjects());
end;
```



Copiando registros de um banco de dados corrompido



FERNANDO SARTURI PRASS
(fernando@dotbr.com.br)
Mestre em Ciência da Computação pela UFSC. Professor da Universidade Luterana do Brasil (ULBRA) nos campus de Santa Maria e Cachoeira do Sul. Sócio-diretor da dotBR Soluções em TI (*www.dotbr.com.br*), empresa que presta serviços de desenvolvimento de sistemas e de consultoria em Bancos de Dados e Metodologias de Desenvolvimento.

Recentemente a empresa na qual sou consultor foi contratada pela prefeitura de uma grande capital para prestar treinamento e consultoria em Firebird. O principal trabalho que deveria ser realizado durante a consultoria era o de otimização do banco de dados da instituição.

Para ter uma idéia do tamanho do desafio, o banco tinha aproximadamente 120 milhões de registros armazenados em pouco mais de 30 tabelas, o que gera um arquivo com 19 GB. Como se o volume de informação já não fosse um desafio suficientemente grande, a análise do banco de dados mostrou que o mesmo estava corrompido.

Este artigo descreve o trabalho que foi realizado para que fosse possível copiar as informações do banco de dados corrompido. Antes disso, mostra rapidamente qual o ambiente encontrado e os principais passos que foram tomados antes da cópia dos dados.

Avaliação da estrutura

O primeiro passo a ser feito sempre que se deseja melhorar o desempenho de uma aplicação é avaliar o servidor (*hardware*) em que ela está rodando. Muitas vezes comprar um novo equipamento, ou melhorar o existente, não gera o ganho de desempenho esperado, fazendo com que o gasto não se justifique.

O servidor usado pelo cliente tinha dois processadores, 2 GB de memória RAM, discos rígidos de 10 mil RPMs e sistema operacional Windows 2000 Server. Ou seja, o equipamento era de alta qualidade e havia pouco o que melhorar.

Avaliação das configurações do banco de dados

A versão instalada do Firebird era a 1.5, rodando como *SuperServer*. Embora muitos ainda considerem a versão *Classic* a ideal, testes realizados demonstraram que a arquitetura *SuperServer* é bastante estável em servidores de alta qualidade. Esse

caso é mais um exemplo.

Avaliamos também a forma de gravação dos dados. O banco de dados estava configurado para usar *cache* de gravação do tipo *write throug* (padrão de instalação do Firebird em plataforma Windows), onde cada operação de escrita é imediatamente passada para que o SO a processe. Em outras palavras, a opção *Enable Forced Writes* estava habilitada, fazendo com que o banco gravasse em disco instantaneamente toda atualização feita nos dados, tornando assim as operações de escrita mais lentas.

Outro ponto avaliado foi o tamanho do *cache* das páginas de dados e de índices. Constatou-se que o tamanho era de 120 MB, dada a quantidade de memória disponível e também o número de usuários conectados, avaliamos que o tamanho estava apropriado.

O penúltimo ponto analisado foi o tamanho das páginas de dados e de índices. Normalmente indicamos aos clientes páginas de dados com 8 Kb. Entretanto, devido ao tamanho do arquivo FDB, optamos por usar páginas de 16 Kb. A lógica é simples, com páginas maiores teremos menos fragmentos de registros divididos, uma melhor utilização dos índices e operações de I/O mais contínuas. Como a mudança no tamanho exige um *backup* e um *restore*, optou-se por fazer essa mudança no final.

Por último foram verificadas as estatísticas das tabelas e índices. Foi verificado que as tabelas mais importantes, aquelas que tinham mais de 20 milhões de registros, estavam fragmentadas, ou seja, não estavam gravadas de maneira sequencial no disco, fazendo com que sejam necessárias várias leituras no disco para obter a informação, causando lentidão.

Uma vez constatada que a mudança no tamanho das páginas de dados e índices e a desfragmentação poderiam melhorar significativamente o desempenho da aplicação, passou-se para a realização de um *backup* para um posterior *restore*. Aí começaram os problemas, pois o banco estava corrompido.

Banco de dados corrompido

Para um banco de dados de 19 GB e com mais de 120 milhões de registros, a situação encontrada foi considerada bastante tranquila:

- Dois registros corrompidos, o que significa dizer que dois registros foram perdidos;
- Duas páginas de dados corrompidas: significa dizer que vários registros foram perdidos, já que uma página de dados armazena *n* registros (o valor de *n* varia de tabela para tabela);
- 161 páginas de índices corrompidas: significa que, embora os dados estejam gravados corretamente no banco de dados, uma consulta pode não encontrar as informações caso a parte corrompida do índice esteja apontando justamente para elas (se o índice for usado na consulta, claro).

As análises realizadas revelaram que a corrupção se deu em apenas uma tabela, que armazenava cerca de 21 milhões de registros. Levando em conta a ocupação média dessa tabela (80%) e o número de registros que podem ser gravados numa página de dados dela (101), estimaram-se que poderiam ter sido perdidos aproximadamente 163 registros, o que poderia causar a não emissão de determinados relatórios, ou ainda pior, a emissão de relatórios com dados incompletos (errados).

Foram feitas diversas tentativas com *softwares* de terceiros para corrigir o banco de dados. Apesar da qualidade dos *softwares* utilizados, após horas de processamento infelizmente todos falharam. O que fazer nessa situação, uma vez que todos os recursos de correção haviam sido tentados sem sucesso?

Resolvendo o problema

A solução encontrada para resolver o problema foi criar um novo banco de dados e migrar os dados (registro a registro) do banco atual para o novo banco, com estrutura idêntica, porém intacto.

A idéia é relativamente simples, desenvolver um programa que, primeiramente, extraia as informações do banco e grave as mesmas em arquivos textos. Posteriormente, após a cópia de todos os registros, transferir os dados armazenados nos arquivos para a nova base de dados.

Após a conclusão do processo de cópia, comparar os dois bancos de dados a fim de verificar se existe correspondência entre o número de registros de cada uma das tabelas (o fluxo do sistema pode ser visto na **Figura 1**). Tudo muito fácil, se não fosse o enorme volume de informações.

Estimou-se, através de testes de I/O no servidor, que o tempo necessário para a realização de todo o processo era de aproximadamente 70 horas. Devido ao tempo demasiadamente longo, o uso de arquivos textos intermediários foi necessário para evitar que o processo tivesse que ser recomeçado do zero caso alguma falha, como falta de energia, ocorresse durante o processo de extração ou de cópia dos dados.

Construindo o sistema

Antes de construir o sistema, uma convenção usada aqui no artigo: sempre que um componente que foi adicionado ao pro-

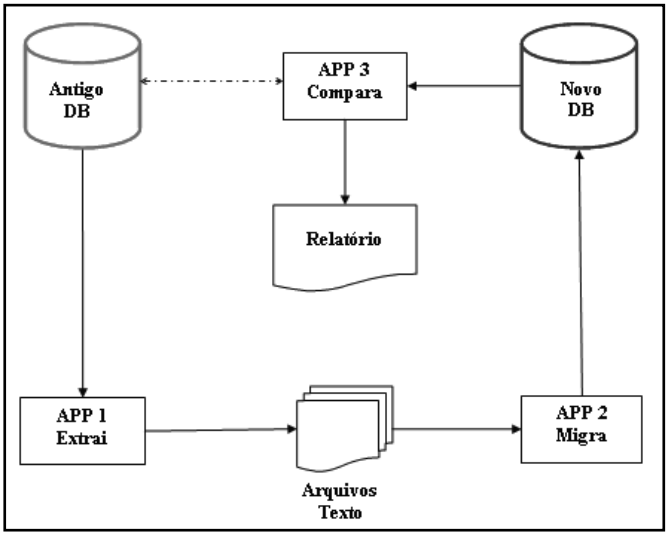


Figura 1. Fluxo geral do processo de migração

jeto tiver seu nome alterado, o novo nome estará entre parênteses logo após o nome da classe. Por exemplo, onde está escrito "adicionar um *Edit* ("edt_teste")" ler como "adicionar um *Edit* e renomear para "edt_teste"".

Inicie um novo projeto, renomeie o formulário para "frm_principal" e adicione um *Edit* ("edt_pasta_arquivos") que será usado para informar o caminho em que serão armazenados os arquivos texto gerados.

Acrescente também dois componentes *ListBox* ("lb_tabelas_origem" e "lb_tabelas_destino"), um para mostrar todas as tabelas e o outro para as tabelas selecionadas pelo usuário. Adicione quatro *SpeedButtons* ("btn_add", "btn_del", "btn_add_todos" e "btn_del_todos") para a inclusão/exclusão das tabelas e dois *BitBtn* ("btn_migrar_extrair" e "btn_fechar"). O código dos botões (exceto do *btn_migrar_extrair*) está na **Listagem 1**. A interface da tela pode ser vista na **Figura 2**.

Adicione um Data Module ("DM") ao projeto e acrescente ao mesmo dois componentes *IBDatabase* ("db_origem" e "db_destino") e dois *IBTransactions* ("trnsc_origem" e "trnsc_destino"). Faça as devidas configurações e aponte os componentes para os seus respectivos bancos de dados. Adicione um *IBQuery* ("qry") apontando para o *db_origem* e um *IBScript* ("script") apontando para *db_destino*.

> **Observação:** *O autor optou por usar IBX, mas se o leitor preferir pode usar outras bibliotecas, como dbExpress ou IBO.*

Extração/migração dos dados

Acrescente mais um formulário ("frm_execucao") ao programa. Esse formulário fará a extração dos dados, migração, comparação e ainda mostrará o *log* das ações do projeto. Adicione um *Memo* ("log") e dois *BitBtns* ("btn_executar" e "btn_fechar") para executar a operação e fechar o formulário após finalizar o processo (**Figura 3**).

O algoritmo da extração é, resumidamente, o seguinte: para cada uma das tabelas selecionadas pelo usuário o sistema deve obter o nome dos seus campos, selecionar todos os registros da mesma e depois montar uma instrução de INSERT para cada um dos seus registros.

A seguir deve armazenar os INSERTs no(s) arquivo(s) texto. A extração dos dados está em *ExtraiDados* do formulário, cujo código pode ser visto na **Listagem 2**.

Sobre o código: cada arquivo terá no máximo 10.000 registros, toda vez que algo importante acontece, como a criação de um novo arquivo, o *log* é atualizado, isso é feito através do *AtualizaLog* (implementando mais adiante).

Ao ler os dados e montar o INSERT é chamada a função *TransformaDado*, que como o próprio nome sugere, transforma a informação armazenada no campo. Essa transformação

Listagem 1. Código para os botões de inclusão/exclusão de tabelas

```
procedure Tfrm_tabelas.btn_addClick(Sender: TObject);
begin
  if lb_tabelas_origem.ItemIndex > -1 then
    if lb_tabelas_destino.Items.IndexOf(
      lb_tabelas_origem.Items[
      lb_tabelas_origem.ItemIndex]) = -1 then
        lb_tabelas_destino.Items.Add(
          lb_tabelas_origem.Items[
          lb_tabelas_origem.ItemIndex]);
end;

procedure Tfrm_tabelas.btn_delClick(Sender: TObject);
begin
  if lb_tabelas_destino.ItemIndex > -1 then
    lb_tabelas_destino.Items.Delete(
      lb_tabelas_destino.ItemIndex);
end;

procedure Tfrm_tabelas.btn_add_todosClick(
  Sender: TObject);
begin
  lb_tabelas_destino.Items.Clear;
  lb_tabelas_destino.Items :=
    lb_tabelas_origem.Items;
end;

procedure Tfrm_tabelas.btn_del_todosClick(
  Sender: TObject);
begin
  lb_tabelas_destino.Items.Clear;
end;

procedure Tfrm_tabelas.btn_fecharClick(
  Sender: TObject);
begin
  Application.Terminate;
end;
```

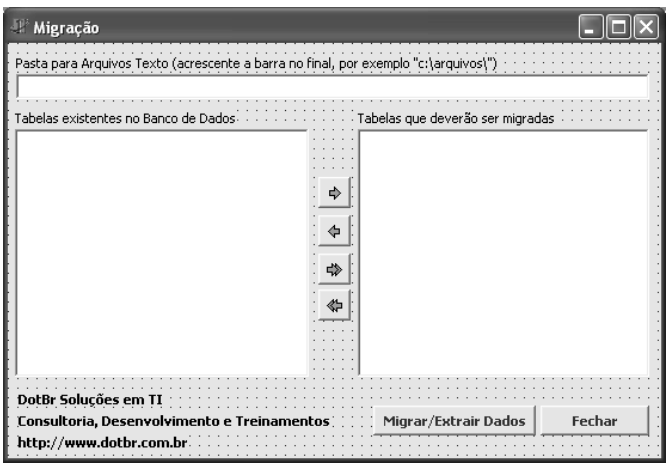


Figura 2. Seleção das tabelas que serão migradas

Figura 3. Acompanhamento do processamento

é necessária em alguns casos, como por exemplo, em campos nulos (precisa colocar a palavra NULL no INSERT) ou campos do tipo *Date* (deve estar com o formato YYYY-MM-DD). O código dessas funções pode ser visto na **Listagem 3**.

> **Atenção: *AtualizaLog*** *deve estar no* ***frm_execucao*** *e os demais procedimentos presentes na* **Listagem 3** *devem estar no Data Module.*

Já o algoritmo da migração dos dados é o seguinte: o sistema seleciona o(s) arquivo(s) texto(s) e, para cada um dos arquivos, abre uma transação e executa os INSERT existentes no arquivo.

Listagem 2. Código para extração de dados das tabelas

```
procedure Tfrm_execucao.ExtraiDados;
var
  fArquivo: TStrings;
  fInsert, fNomeArquivo: string;
  fNrArquivo, fCont1, fCont2, fNrRegistros: integer;

begin
  fArquivo := TStringList.Create;
  AtualizaLog('** INICIANDO EXTRAÇÃO DE REGISTROS **');
  for fCont1:= 0 to
    frm_principal.lb_tabelas_destino.Items.Count - 1 do

  begin
    fNrArquivo := 0;
    fNrRegistros := 0;
    AtualizaLog('Tabela ' +
      frm_principal.lb_tabelas_destino.Items[fCont1] +
      ': obtendo dados');
    dm.qry.SQL.clear;
    dm.qry.SQL.Add('select * from '+
      frm_principal.lb_tabelas_destino.Items[fCont1]);
    dm.qry.open;

    while not dm.qry.Eof do
    begin
      fInsert := 'insert into ' +
        frm_principal.lb_tabelas_destino.Items[
        fCont1] + ' values (';

      for fCont2 := 0 to dm.qry.Fields.Count-1 do
        if fCont2 <> dm.qry.Fields.Count-1 then
          fInsert := fInsert + dm.TransformaDado(
            dm.qry.Fields[fCont2]) + ','
        else
          fInsert := fInsert + dm.TransformaDado(
            dm.qry.Fields[fCont2]) + ');';
      fArquivo.Add(fInsert);
      inc(fNrRegistros);
      dm.qry.Next;

      if ((fNrRegistros/10000) = int(
        fNrRegistros/10000)) or (dm.qry.Eof) then
      begin
        (fArquivo as TStringList).Sort;
        Inc(fNrArquivo);

        fNomeArquivo := UpperCase(
          frm_principal.lb_tabelas_destino.Items[
          fCont1] + '_' + IntToStr(fNrArquivo) +
          '.txt');

        fArquivo.SaveToFile(
          frm_principal.edt_pasta_arquivos.Text +
          '\'+fNomeArquivo);
        AtualizaLog('Arquivo ' + fNomeArquivo +
          ': exportado com sucesso');

        fArquivo.Clear;
      end;

    end;
  end;
  AtualizaLog('*** FIM DA EXTRAÇÃO DE REGISTROS ***');
  AtualizaLog('');
  dm.qry.Close;
  dm.trnsc_origem.Commit;
end;
```

Listagem 3. Atualização do log e transformação de dados

```
procedure Tfrm_execucao.AtualizaLog(pMensagem: string);
begin

  if pMensagem <> '' then
  begin
    log.Lines.Add(DateTimeToStr(now) + ': ' + pMensagem);
    log.Lines.SaveToFile(
      frm_principal.edt_pasta_arquivos.Text + '\log.log');
  end
  else
    log.Lines.Add('');

end;

{ Declarar variável na seção public do DataModule }
fErroExecucao : Boolean;

...
function tdm.MudaCaracter(pString, pCaracterVelho,
  pCaracterNovo: string) : string;
var
  fCont: integer;
begin

  if length(pString) = 0 then
    Result := pString
  else
  begin
    fcont := pos(pCaracterVelho, pString);

    while fcont > 0 do
    begin
      pstring:= copy(pString, 1, fcont-1) +
        pCaracterNovo + copy(pString, fcont + length(
        pCaracterVelho), 3000);
      fCont := pos(pCaracterVelho, pString);
    end;
    Result := pString;
  end;

end;

function Tdm.InverteData(pField: TField): string;
var
  d, m, a: word;
begin
  case pField.DataType of
    ftDate:
      begin
        DecodeDate(pField.AsDateTime, a, m, d);
        Result := IntToStr(a) + '/' + IntToStr(m) +
          '/' + IntToStr(d);
      end;

    ftDateTime:
      begin
        DecodeDate(StrToDate(copy(pField.AsString, 1,
          10)), a, m, d);
        Result := IntToStr(a) + '/' + IntToStr(m) +
          '/' + IntToStr(d) + copy(pField.AsString,
          11, 9);
      end;
  end;
end;

function Tdm.TransformaDado(pField: TField): string;
begin

  if pField.IsNull then
    Result:= 'null'
  else
  begin
    case pField.DataType of
      ftFloat: Result := dm.MudaCaracter(pField.AsString, ',' ,'.');
      ftString:

        begin
          Result := dm.MudaCaracter(pField.AsString, ';', ':');
          Result := dm.MudaCaracter(Result, '/*', '|*');
          Result := dm.MudaCaracter(Result, '--', '- ');
          Result := QuotedStr(dm.MudaCaracter(Result, chr(39),' '));
          { chr(39) = ' (apóstofro) }
        end;

      ftDateTime,ftDate: Result := QuotedStr(dm.InverteData(pField));
      else

    Result := pField.AsString;

    end;

  end;
end;
```

Se não houver nenhum erro, a transação recebe um *Commit* e o arquivo é renomeado para OK_ANTIGO_NOME, do contrário recebe um *Rollback* e o arquivo é renomeado para ERRO_ANTIGO_NOME. O código da *MigraDados* está na **Listagem 4**.

Listagem 4. Código para migração dos dados

```
procedure Tfrm_execucao.MigraDados;
var
  fNrRegistros, fCont: integer;
  fSearchResult: TSearchRec;
  fArquivo: TStrings;
begin
  fArquivo := TStringList.Create;
  { pega a lista de arquivos na pasta indicada }
  if FindFirst(frm_principal.edt_pasta_arquivos.Text +
    '*.txt', faAnyFile, fSearchResult) = 0 then
  begin
    repeat
      fArquivo.Add(fSearchResult.Name);
    until FindNext(fSearchResult) <> 0;
    FindClose(fSearchResult);
  end;
  dm.trnsc_destino.StartTransaction;
  AtualizaLog('** INICIANDO MIGRAÇÃO DE REGISTROS **');
  for fCont:=0 to fArquivo.Count-1 do
  begin
    AtualizaLog('Migrando arquivo ' + fArquivo[fCont]);
    dm.script.SQL.LoadFromFile(
      frm_principal.edt_pasta_arquivos.Text +
      fArquivo[fCont]);
    dm.fErroExecucao := False;
    dm.script.ExecSQL;
    if not dm.fErroExecucao then
    begin
      dm.script.Transaction.CommitRetaining;
      fNrRegistros := fNrRegistros +
        dm.script.SQL.Count;
      AtualizaLog('Arquivo ' + fArquivo[fCont] +
        ' - SUCESSO');
      RenameFile(fArquivo[fCont],
        frm_principal.edt_pasta_arquivos.Text +
        '\OK_' + fArquivo[fCont])
    end
    else
    begin
      dm.script.Transaction.RollbackRetaining;
      AtualizaLog('Arquivo ' + fArquivo[fCont] + ' - FALHOU');
      RenameFile(fArquivo[fCont],
        frm_principal.edt_pasta_arquivos.Text + '\ERRO_' + fArquivo[fCont])
    end;
  end;
  AtualizaLog('*** FIM DA MIGRAÇÃO DE REGISTROS ***');
  AtualizaLog('');
  dm.trnsc_destino.Commit;
end;
```

Por último, o mais simples dos algoritmos, o que compara o número de registro de cada uma das tabelas (*ComparaDados*). Para tanto, o sistema deve obter o número de registros da tabela no banco de dados original e no novo banco de dados (basta usar o comando *select count(*) from nome_tabela*) e comparar os números.

Se forem diferentes deve adicionar uma linha no *log* com o nome da tabela cujo número de registros não bate. Como esse código é bem mais simples, deixarei a cargo do leitor a sua implementação.

Para finalizar os trabalhos na *unit* do *frm_execucao*, inclua no evento *OnClick* do *btn_executar* uma chamada para *ExtraiDados*, *MigraDados* e *ComparaDados* e uma chamada ao *Close* do *btn_fechar*.

De volta ao *frm_principal*, acrescente uma chamada ao evento *GetTableNames* (traz o nome de todas as tabelas existentes no banco) do *db_origem* no evento *OnCreate* do formulário:

```
dm.db_origem.GetTableNames(lb_tabelas_origem.Items);
```

E no evento *OnClick* do *btn_migrar_extrair* coloque o código:

```
frm_execucao := Tfrm_execucao.Create(self);
try
  frm_execucao.ShowModal;
finally
  frm_execucao.Free;
end;
```

Ganho de desempenho obtido

O trabalho de otimização visava atender principalmente duas coisas: melhorar a importação de arquivos (dados em arquivos textos que são importados para o sistema) e diminuir o tempo de emissão dos relatórios gerenciais. A **Tabela 1** mostra o ganho de desempenho obtido na importação dos arquivos texto.

Tamanho do Arquivo	Tempo Antes (hh:mm)	Tempo Depois (hh:mm)	Redução do Tempo
1.4 MB	02:06	00:34	73,1 %
3.2 MB	03:56	01:17	67,4 %
5.8 MB	06:51	02:41	60,8 %

Tabela 1. Ganho de desempenho na importação dos dados

Com o banco de dados corrompido e sem as mudanças de configuração apontadas no começo do artigo, eram processados aproximadamente três arquivos numa noite, após o trabalho esse número passou para oito.

Já a **Tabela 2** mostra que o ganho de desempenho obtido na emissão de alguns dos relatórios mais usados no sistema foi de aproximadamente 75% (os nomes dos relatórios foram propositalmente ocultados).

Relatório	Tempo Antes (hh:mm:ss)	Tempo Depois (hh:mm:ss)	Redução do Tempo
Relatório 1	00:14:12	00:03:19	76,6 %
Relatório 2	01:38:16	00:26:01	73,5 %
Relatório 3	15:36:19	04:04:37	73,8 %

Tabela 2. Ganho de desempenho na emissão dos relatórios

Conclusão

Muitas vezes, mesmo com o auxílio de ferramentas de reconhecida qualidade, não é possível reparar um banco de dados corrompido. Nesses casos, têm-se duas opções: restaurar um *backup* e perder tudo o que foi adicionado/alterado/excluído depois do *backup* ou migrar todos os dados, registro a registro, como nesse caso. ■

Seja um assinante Gold!

Com a assinatura Gold você já pode consultar online todos os artigos publicados na sua revista desde a edição nº 1.

DevMedia GROUP

Saiba Mais! Acesse:
www.devmedia.com.br/assgold

Para mais informações:
www.devmedia.com.br/central

Assinatura Gold

Atenção: Já encontram-se disponíveis as assinaturas GOLD das revistas WebMobile e .net Magazine.

Como trabalhar com imagens em aplicações ASP.NET

FABRÍCIO DESBESSEL
(fabricio@fabricio.pro.br)
é professor de Linguagem de Programação do Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração está sempre disposto a provar que com o Delphi sempre teremos a melhor solução. Site *www.fabricio.pro.br*.

Em aplicações Web, é comum a necessidade de trabalhar com imagens. Especialmente com fotos e com serviços de redimensionamento e aplicações de marcas, juntando diferentes imagens. Isso tudo pode ser feito no Delphi for .NET através do namespace *System.Drawing.Imaging* que fornece funcionalidades avançadas de imagem (GDI+). A funcionalidade básica dos gráficos é fornecida pelo namespace de *System.Drawing*.

Neste artigo, veremos como trabalhar com imagens em aplicações ASP.NET, usando os namespaces acima descritos. Veremos com redimensionar uma imagem, além de inserir uma marca d'água.

Respondendo uma imagem

Inicialmente vamos criar uma aplicação ASP.NET que responderá uma imagem, conforme um campo de consulta (*QueryString*), informado pelo usuário, através do link. Crie uma nova aplicação ASP.NET através do menu *File>ASP.NET Web Application - Delphi For .NET*. Defina um nome para a aplicação e lembre-se de estar com o IIS rodando para que o Delphi crie a pasta virtual.

Para este exemplo nomeie a aplicação como "Foto01". Acesse o *Page_Load* e digite o código da **Listagem 1**. O código está comentado, de forma que fica fácil o entendimento.

Listagem 1. Código do Page_Load

```
var
  Imagem: System.Drawing.Image;
begin
if System.IO.File.Exists(Server.MapPath(
    '\Foto01\fotos\')+Request.QueryString['f']) then
  begin
    Imagem := System.Drawing.Image.FromFile(
      Server.MapPath('\Foto01\fotos\')+
      Request.QueryString['f']);
    Response.ContentType := 'image/jpeg';
    Imagem.Save(Response.OutputStream,
      System.Drawing.Imaging.ImageFormat.Jpeg);
    Imagem.Dispose();
  end;
end;
```

Antes de testar crie uma pasta chamada "fotos" dentro da pasta da aplicação (em *C:\Inetpub\wwwroot\Foto01*). Copie algumas fotos para essa pasta e acrescente os namespaces *System.Drawing.Imaging* e *System.IO* na cláusula *uses*.

Compile a aplicação e altere o link informado no *browser*, acrescentando o campo de consulta "f" que informará o arquivo da foto que será respondido (Ex.: http://localhost/Foto01/WebForm1.aspx?f=nomefoto.jpg). Veja na **Figura 1** um exemplo (veja o nome do arquivo na URL).

Explicando o código

O primeiro comando verifica se o arquivo de imagem passado pelo campo de consulta realmente existe. Se existir, a variável *Imagem* será carregada com o arquivo. Define-se o conteúdo de resposta (*image/jpeg*) e reponde-se a imagem para a aplicação.

Redimensionando uma imagem antes de responder

Geralmente precisamos limitar o tamanho da imagem antes de exibi-la, para que o carregamento das mesmas não fique muito longo. O interessante é que utilizando o namespace *System.Drawing.Imaging* pode-se fazer isso de forma rápida e com um bom resultado na diminuição do tamanho do arquivo, agilizando a apresentação da foto no browser.

Crie uma nova aplicação ASP.NET dando o nome de "Foto02". Acesse o *Page_Load* e digite o código da **Listagem 2**.

Listagem 2. Código para redimensionar a imagem

```
var
  imgBase, imgFinal: System.Drawing.Image;
  largura, altura: Double;
  dimensoes: Size;
  grafico: System.drawing.Graphics;
  retangulo: Rectangle;
begin
  if System.IO.File.Exists(Server.MapPath(
    '\Foto02\fotos\')+Request.QueryString['f']) then
  begin
    imgBase := System.Drawing.Image.FromFile(
      Server.MapPath('\Foto02\fotos\')+
      Request.QueryString['f']);
    largura := Convert.ToInt16(
      Request.QueryString['t']);
    if largura = 0 then
    begin
      largura := imgBase.Width;
      altura := imgBase.Height;
    end
    else
      altura := imgBase.Height/(
      imgBase.Width/largura);
    dimensoes:= Size.create(Convert.ToInt32(largura),
      Convert.ToInt32(altura));
    imgFinal := System.Drawing.Bitmap.Create(
      dimensoes.Width, dimensoes.Height,
      imgBase.PixelFormat);
    grafico := System.Drawing.Graphics.FromImage(
      imgFinal);
    retangulo := Rectangle.Create(0, 0,
      dimensoes.Width, dimensoes.Height);
    grafico.DrawImage(imgBase, retangulo);

    Response.ContentType := 'image/jpeg';
    imgFinal.Save(Response.OutputStream,
      System.Drawing.Imaging.ImageFormat.Jpeg);
    imgFinal.Dispose;
    imgBase.Dispose;
  end;
end;
```

Acrescente os namespaces *System.Drawing.Imaging* e *System.IO* na cláusula *uses*. Crie a pasta "fotos" e copie algumas fotos.

Compile a aplicação e modifique o link para: *http://localhost/Foto02/WebForm1.aspx?f=nomefoto.jpg&t=300*, para que a imagem apareça redimensionada no browser (**Figura 2**).

Explicando o código

Passamos também, através de um campo de consulta (*t*), a largura da foto. Assim, foi necessário fazer um cálculo proporcional para que a imagem não perca a relação entre largura e altura. Utilizamos duas imagens: a *base* e a *final* e foi necessário utilizar um gráfico e um retângulo para fazer o redimensionamento da imagem.

Para facilitar também utilizamos uma variável do tipo *Size* que possui as dimensões *Width* e *Height*. Essa opção é interessante porque não é feita somente uma alteração no tamanho apresentado, mas sim, no tamanho do arquivo que será carregado.

Realizou-se alguns testes onde uma imagem de 101 Kb respondida pela aplicação, ainda sem alteração de tamanho, foi respondida com um tamanho de 73,9 Kb sem perder qualidade. Em outro teste solicitou-se a reposta com um tamanho de 100 *pixels* e o arquivo exibido na resposta teve seu tamanho reduzido para 3,46 Kb, apresentando uma boa taxa de compressão.

Colocando uma marca na imagem

Muitos Web Designers perdem um bom tempo modificando fotos para, simplesmente, colocar uma marca da empresa, endereço do site ou *copyright*. No Delphi, podemos sobrepor

Figura 1. Exemplo de como exibir imagens no browser

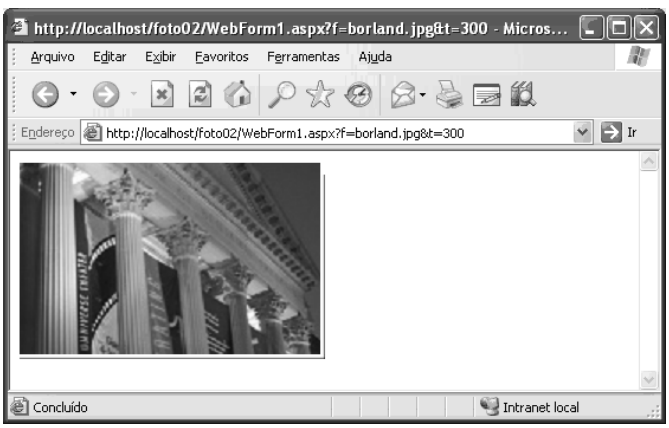


Figura 2. Redimensionando imagens no browser

uma imagem à outra, criando assim um efeito de marca d'água (dependendo do estilo da imagem que será marca).

Basicamente, esse efeito é criado através da sobreposição de imagens e/ou utilização de transparências, portanto, adicione um arquivo chamado *marca.gif* que será sobreposto sobre outra imagem. Crie uma nova aplicação dando o nome de "Foto03". Acesse o *Page_Load* e digite o código contido na **Listagem 3**.

Listagem 3. Código para colocar uma marca na imagem

```
var
  imgBase, imgFinal, imgMarca: System.Drawing.Image;
  largura, larguraMarca, altura, alturaMarca: Double;
  dimensoes: Size;
  grafico: System.drawing.Graphics;
  retangulo, retanguloMarca:  Rectangle;
  inicioX, inicioY: Integer;
begin
  if System.IO.File.Exists(Server.MapPath(
    '\Foto03\fotos\')+Request.QueryString['f']) then
  begin
    imgBase := System.Drawing.Image.FromFile(
      Server.MapPath('\Foto03\fotos\')+
      Request.QueryString['f']);
    largura := imgBase.Width;
    altura := imgBase.Height;
    dimensoes:= Size.create(Convert.ToInt32(largura),
      Convert.ToInt32(altura));
    imgFinal := System.Drawing.Bitmap.Create(
      dimensoes.Width, dimensoes.Height,
      imgBase.PixelFormat);
    grafico := System.Drawing.Graphics.FromImage(
      imgFinal);
    retangulo := Rectangle.Create(0, 0,
      dimensoes.Width, dimensoes.Height);
    grafico.DrawImage(imgBase, retangulo);
    if System.IO.File.Exists(Server.MapPath(
      '\Foto03\fotos\')+'marca.gif') then
    begin
      imgMarca := System.Drawing.Image.FromFile(
        Server.MapPath('\Foto03\fotos\')+'marca.gif');
      larguraMarca := imgMarca.Width;
      alturaMarca := imgMarca.Height;

      if Request.QueryString['p'] = 't' then
      begin
        inicioX := 0;
        inicioY := 0;
      end;
      if Request.QueryString['p'] = 'b' then
      begin
        inicioX := Convert.ToInt32(dimensoes.Width) -
          Convert.ToInt32(larguraMarca);
        inicioY := Convert.ToInt32(dimensoes.Height) -
          Convert.ToInt32(alturamarca);
      end;
      dimensoes.Width := Convert.ToInt32(larguraMarca);
      dimensoes.Height := Convert.ToInt32(alturaMarca);
      retanguloMarca := Rectangle.Create(inicioX,
        inicioY, dimensoes.Width, dimensoes.Height);
      grafico.DrawImage(imgMarca, retanguloMarca);
    end;
    Response.ContentType := 'image/jpeg';
    imgFinal.Save(Response.OutputStream,
      System.Drawing.Imaging.ImageFormat.Jpeg);
    imgFinal.Dispose;
    imgBase.Dispose;
  end;
end;
```

Não esqueça de declarar no *uses* os namespaces *System.Drawing.Imaging* e *System.IO*. Também crie a pasta "fotos" e copie algumas fotos antes de compilar e testar.

Explicando o código

Para sobrepor uma imagem à outra foi necessária a utilização de dois retângulos. Criou-se um campo de consulta (*Query String* chamada *p*) para definir a posição da marca, podendo ser no topo (*t*) ou em baixo (*b*). Se for no topo, a marca será exibida no canto superior esquerdo. Se for embaixo será exibida no canto inferior direito. Para colocar a figura embaixo foi necessário fazer um cálculo para definir a posição inicial de altura e largura (X e Y).

Compile a aplicação e modifique o link para: *http://localhost/*

Figura 3. Imagem com uma marca

Figura 4. Browser exibindo uma imagem redimensionada e com uma marca

Listagem 4. Imagem com marca e tamanho definidos

```
const
  Proporcao = 2;
var
  imgBase, imgFinal, imgMarca: System.Drawing.Image;
  largura, larguraMarca, altura, alturaMarca: Double;
  dimensoes: Size;
  grafico : System.drawing.Graphics;
  retangulo, retanguloMarca :  Rectangle;
  inicioX, inicioY, tamanho : Integer;
begin
  if System.IO.File.Exists(Server.MapPath(
    '\Foto04\fotos\')+Request.QueryString['f']) then
    imgBase := System.Drawing.Image.FromFile(
      Server.MapPath('\Foto04\fotos\')+Request.QueryString['f'])
  else
    imgBase := System.Drawing.Image.FromFile(
      Server.MapPath('\Foto04\fotos\indisponivel.jpg'));
  tamanho := Convert.ToInt32(Request.QueryString['t']);
  if tamanho = 0 then
  begin
    largura := imgBase.Width;
    altura := imgBase.Height;
  end
  else
  begin
    largura := tamanho;
    altura := imgBase.Height/(imgBase.Width/tamanho);
  end;
  dimensoes := Size.create(Convert.ToInt32(largura),
    Convert.ToInt32(altura));
  imgFinal := System.Drawing.Bitmap.Create(
    dimensoes.Width, dimensoes.Height, imgBase.PixelFormat);
  grafico := System.Drawing.Graphics.FromImage(imgFinal);
  retangulo := Rectangle.Create(0, 0, dimensoes.Width,
    dimensoes.Height);
  grafico.DrawImage(imgBase, retangulo);
  if (largura > 100) and (System.IO.File.Exists(
    Server.MapPath('\Foto04\fotos\')+'marca.gif')) then
  begin
    imgMarca := System.Drawing.Image.FromFile(
      Server.MapPath('\Foto04\fotos\')+'marca.gif');
    larguraMarca := dimensoes.Width/proporcao;
    alturaMarca := (imgMarca.Height/imgMarca.Width) *
      (dimensoes.Width/proporcao);
    if Request.QueryString['p'] = 't' then
    begin
      inicioX := 0;
      inicioY := 0;
    end;
    if Request.QueryString['p'] = 'b' then
    begin
      inicioX := Convert.ToInt32(dimensoes.Width) -
        Convert.ToInt32(larguraMarca) - 4;
      inicioY := Convert.ToInt32(dimensoes.Height) -
        Convert.ToInt32(alturamarca) - 4;
    end;
    dimensoes.Width := Convert.ToInt32(larguraMarca);
    dimensoes.Height := Convert.ToInt32(alturaMarca);
    retanguloMarca := Rectangle.Create(inicioX,
      inicioY, dimensoes.Width, dimensoes.Height);
    grafico.DrawImage(imgMarca, retanguloMarca);
  end;
  Response.ContentType := 'image/jpeg';
  imgFinal.Save(Response.OutputStream,
    System.Drawing.Imaging.ImageFormat.Jpeg);
  imgFinal.Dispose;
  imgBase.Dispose;
end;
```

Foto03/WebForm1.aspx?f=nomefoto.jpg&p=t, para que a imagem apareça com uma marca no browser na parte superior (**Figura 3**).

Incluindo uma marca com tamanho proporcional ao redimensionamento

Agora juntaremos todos os exemplos vistos até então, possibilitando ao usuário a definição do tamanho da imagem que será exibida, adicionando uma marca (texto) que terá um tamanho proporcional ao tamanho final da imagem e ainda escolhendo a posição dessa marca.

Se a imagem não for encontrada, será respondida uma imagem (o arquivo está para download). Para o arquivo de marca, criou-se uma imagem contendo o texto "ClubeDelphi", que foi salva no formato GIF, com fundo transparente (chamada *marca.gif*).

Vamos criar uma nova aplicação dando o nome de "Foto04". Acesse o *Page_Load* e digite o código contido na **Listagem 4**.

Novamente, não esqueça de declarar no *uses* os namespaces *System.Drawing.Imaging* e *System.IO* e também criar a pasta "fotos" e copiar algumas fotos antes de compilar e testar.

Explicando o código

Definiu-se uma constante para estabelecer a proporção do tamanho da marca levando em conta o tamanho da imagem definido pelo campo de consulta (*t*) do link. Com isso foi necessário fazer o cálculo do tamanho da marca levando em conta a proporção definida na constante. A marca criada para o exemplo na **Figura 4** teve seu tamanho definido em 1157 x 44 *pixels,* salva no formato GIF, com fundo transparente.

Conclusão

Apresentamos mais uma facilidade do ASP.NET, onde podermos criar, muito rapidamente, uma aplicação que trabalhe com imagens para a internet. Exemplos como álbuns de fotos, sistemas de vendas de veículos, imobiliárias etc., são ótimos candidatos a usufruir das técnicas aqui demonstradas. ■

Um guia de como e quando utilizar Generators no Firebird

FRANK INGERMANN

Este artigo explica o que são os *Generators* do Firebird e como e por que devemos utilizá-los. Representa uma tentativa de coletar todas as informações relevantes a respeito de *Generators* em um único documento.

Quem deve ler este artigo?

Leia este artigo se você:

- Não está familiarizado com o conceito de Generators;
- Tem dúvidas de como utilizá-lo;
- Quer criar um campo ID no estilo "auto-incremento" como visto em outros banco de dados;
- Está procurando exemplos de como utilizar Generatos para incrementar campos;

O que é um generator?

Imaginemos um *Generator* como um contador de números inteiros residente em um banco de dados Firebird. Poderemos criar um *Generator* e ao mesmo tempo atribuir-lhe um nome, com a seguinte declaração:

```
CREATE GENERATOR GenTest;
```

A seguir podemos obter o seu valor atual, incrementá-lo ou decrementá-lo como faríamos com um *var i: integer* em Delphi, porém, não é fácil configurá-lo diretamente com certo valor de um modo previsível e a seguir obter o mesmo valor. Ele encontra-se no banco de dados, porém, fora do controle da transação.

O que é uma Sequence?

O termo *Sequence* é a designação oficial SQL para o que o Firebird chama de *Generator*. Como a equipe do Firebird está constantemente se esforçando para melhor se enquadrar no padrão SQL, o termo SEQUENCE pode ser utilizado como sinônimo de GENERATOR no Firebird 2 e superior. De fato, é recomendável a utilização da sintaxe SEQUENCE, quando formos escrever novo código.

Embora a palavra *Sequence* ponha a ênfase na série de valores gerados, ao passo que *Generator* parece referir-se principalmente ao mecanismo fonte que produz esses valores, não existe em absoluto nenhuma diferença entre um *Generator* e uma *Sequence*. São apenas duas palavras para nos referirmos ao mesmo recurso do banco de dados.

Podemos criar um *Generator* e acessá-lo utilizando a sintaxe de *Sequence* e vice-versa. A seguir, temos a sintaxe preferencial para criarmos um *Generator*/*Sequence* no Firebird 2:

```
CREATE SEQUENCE SeqTest;
```

RDB$GENERATOR_NAME	RDB$GENERATOR_ID	RDB$SYSTEM_FLAG
RDB$SECURITY_CLASS	1	1
SQL$DEFAULT	2	1
RDB$PROCEDURES	3	1
RDB$EXCEPTIONS	4	1
RDB$CONSTRAINT_NAME	5	1
RDB$FIELD_NAME	6	1
RDB$INDEX_NAME	7	1
RDB$TRIGGER_NAME	8	1
MY_OWN_GENERATOR	9	NULL

Tabela 1. Estrutura da tabela RDB$GENERATORS

Onde são armazenados os Generators?

As declarações dos *Generators* são armazenadas na tabela de sistema RDB$GENERATORS. Os seus valores, no entanto, são armazenados em páginas reservadas especiais dentro do banco de dados. Nunca teremos acesso a esses valores diretamente, e sim por meio de funções e declarações embutidas que serão abordadas mais adiante.

> **Obs.:** *As informações fornecidas nessa seção têm objetivos educacionais somente. Como regra geral, não devemos "mexer" com as tabelas do sistema. Não devemos tentar criar ou alterar* ***Generators*** *escrevendo diretamente na tabela RDB$GENERATORS (o acesso mediante declarações SELECT, no entanto, não tem nenhuma contra-indicação).*

A estrutura da tabela de sistema RDB$GENERATORS é a seguinte:

```
RDB$GENERATOR_NAME CHAR(31)
RDB$GENERATOR_ID SMALLINT
RDB$SYSTEM_FLAG SMALLINT
```

E ainda, no Firebird 2.0:

```
RDB$DESCRIPTION BLOB subtype TEXT
```

Observar que o GENERATOR_ID é, como o nome diz, um identificador para cada *Generator* e não o seu valor. Também, não devemos permitir que nossas aplicações armazenem o identificador do generator para utilização posterior. Além de não fazer sentido, o ID pode ser modificado após um ciclo de backup/restore. O SYSTEM_FLAG é 1 para *Generators* utilizados internamente pelo engine do servidor e NULL ou 0 para aqueles criados pelo usuário.

Agora daremos uma olhada na tabela RDB$GENERATORS, aqui com um simples *Generator* auto-definido, conforme vemos na **Tabela 1**.

Notas para o Firebird 2:

• No Firebird 2 foi introduzido um *Generator* de sistema, chamado RDB$BACKUP_HISTORY. É utilizado para a nova funcionalidade do NBackup;

• Embora a sintaxe SEQUENCE seja preferida, a tabela de sistema RDB$GENERATORS e as suas colunas não foram renomeadas no Firebird 2.

Qual é o valor máximo de um Generator?

Os *Generators* armazenam e retornam valores de 64 bits em todas as versões do Firebird. Isso nos dá uma variação de valor de: -2^{63}.. 2^{63-1} ou -9.223.372.036.854.775.808.. 9.223.372.036.854.775.807

Portanto, se utilizamos um *Generator* com o valor inicial 0 para alimentar uma coluna NUMERIC(18) ou BIGINT (ambos tipos representam números inteiros de 64 bits) e a seguir inserirmos 1000 linhas por segundo (o que já é uma frequência considerável), levaria aproximadamente 300 milhões de anos (!) até atingirmos o valor máximo. Como é bastante improvável que o ser humano ainda ande neste planeta até lá (e ainda estiver utilizando bancos de dados Firebird), não é nada com o que devamos nos preocupar.

No entanto, um alerta. O Firebird fala dois dialetos SQL: o dialeto 1 e o dialeto 3. Novos bancos de dados sempre devem ser criados com o dialeto 3, que é mais poderoso em vários aspectos. O dialeto 1 é para compatibilidade, deve ser utilizado somente para bancos de dados legados, que foram criados sob o InterBase 5.6 ou anterior.

Uma das diferenças entre os dois é que o dialeto 1 não tem disponível nenhum tipo numérico nativo de 64. As colunas NUMERIC(18), por exemplo, são totalmente armazenadas com DOUBLE PRECISION, que é do tipo ponto flutuante. O maior número do tipo inteiro no dialeto 1 é o INTEGER de 32 bits.

Tanto no dialeto 1 quanto no dialeto 3, os generators são de 64 bits. Porém, se atribuirmos os valores gerados a uma coluna INTEGER em um banco de dados gerado com o dialeto 1, esses valores serão truncados para os 32 bits de mais baixa ordem, produzindo uma variação eficaz de:

-2^{31}.. 2^{31-1} ou -2.147.483.648.. 2.147.483.647

Embora o próprio *Generator* continue gerando valores de 2.147.483.647 a 2.147.483.648, o valor truncado ficaria em torno deste ponto, dando a impressão de um *Generator* de 32 bits. Na situação descrita anteriormente, com 1000 inserções por segundo, a coluna alimentada pelo *Generator* agora iria estourar após 25 *dias* (!!!), o que de fato é algo com o que se preocupar.

O valor 2^{31} é grande, porém, novamente nesse caso, não muito preocupante dependendo da situação.

> **Obs.:** *No dialeto 3, se atribuirmos valores* ***Generator*** *a*

um campo INTEGER, tudo funcionará bem enquanto os valores permanecerem dentro da faixa de 32 bits. No entanto, assim que a faixa for excedida, obteremos um erro de overflow. O dialeto 3 é muito mais preciso na verificação de faixa do que o dialeto 1!

Dialetos cliente e valores de Generator

Os clientes que falam com um servidor Firebird, podem configurar o seu dialeto em 1 ou 3, independentemente do banco de dados que estejam conectados. É o dialeto do cliente e não o dialeto do banco de dados, que determina o modo como o Firebird passa valores de *Generators* para o cliente:

• Se o dialeto cliente for 1, o servidor retorna valores como números inteiros de 32 bits truncados para o cliente. No entanto, dentro do banco de dados os valores permanecem com 64 bits e não estouram após atingir 2^{31-1} (embora possam ter esse aspecto para o cliente). Isso é verdadeiro tanto para bancos de dados dialeto 1 quanto para o dialeto 3;

• Se o dialeto cliente for 3, o servidor passa o valor de 64 bits completo para o cliente.

Quantos Generators há disponíveis em um banco de dados?

Desde o Firebird versão 1.0, o número de *Generators* que poderemos ter em um único banco de dados é limitado somente pelo máximo ID designado na tabela do sistema RDB$GENERATORS. Sendo um SMALLINT, o valor máximo será de 2^{15-1} ou 32767.

O primeiro ID é sempre 1, portanto, o número total de *Generators* não pode exceder 32767. Como discutido antes, existem 8 ou 9 *Generators* de sistema no banco de dados, deixando espaço para utilização pelo usuário de no mínimo 32758 *Generators*.

Isso deve ser amplo o bastante para atender às necessidades de qualquer aplicação prática e desde que o desempenho não é afetado pela quantidade de *Generators* declarados, estamos livres para utilizar tantos *Generators* quantos sejam necessários.

Versões mais antigas do InterBase e do Firebird

Nas versões anteriores à versão 1.0 do Firebird, bem como no InterBase, somente uma página do banco de dados era utilizada para armazenamento dos valores de *Generators*. Por isso, o número de *Generators* disponíveis estava limitado pelo tamanho da página do banco de dados.

A **Tabela 2** a seguir, demonstra quantos *Generators* (incluindo os *Generators* de sistema) podemos ter nas várias versões do InterBase e do Firebird (agradecemos ao Paul Reeves por ter nos fornecido as informações iniciais):

Em versões do InterBase anteriores à 6, os *Generators* tinham somente 32 bits de comprimento. Isso explica por que essas versões mais antigas podiam mal armazenar duas vezes o número de *Generators* no mesmo tamanho de página.

Versão	Tamanho da Página			
	1 K	2 K	4 K	8 K
InterBase <v.6	247	503	1015	2039
InterBase 6 e primeiras Firebird pré-1.0	123	251	507	1019
Todas as versões posteriores do Firebird	32767			

Tabela 2. Quantidade de Generators do banco de dados

Generators e transações

Como já foi dito, os *Generators* existem fora do controle das transações. Isso significa simplesmente que não podemos, com segurança, fazer o *rollback* dos *Generators* dentro de uma transação. Outras transações podem estar executando simultaneamente e modificando o valor enquanto a nossa transação está rodando. Portanto, uma vez que tivermos solicitado um valor de *Generator*, devemos considerá-lo como "não mais utilizável".

Quando iniciamos uma transação e a seguir chamamos um *Generator* e obtemos um valor de, digamos 5, esse *Generator* permanecerá com esse valor mesmo se fizermos o *rollback* da transação (!). Nem vale a pena pensar em alguma coisa como "tudo bem, quando acontecer o *rollback*, podemos posteriormente apenas fazer *GEN_ID (mygen,-1)* para diminuí-lo para 4".

Isso pode funcionar na maioria das vezes, porém, é perigoso, porque outras transações simultâneas podem ter modificado o valor no meio tempo. Pela mesma razão, não faz sentido obtermos o valor atual com *GEN_ID (mygen, 0)* e a seguir incrementar o valor no lado cliente.

Declarações SQL para Generators

O nome de um *Generator* deve ser um meta-identificador de banco de dados habitual: máximo de 31 caracteres, nenhum caractere especial exceto o underline "_" (a menos que utilizemos identificadores entre aspas duplas).

Os comandos e declarações SQL que se aplicam a *Generators*, estão enumerados a seguir:

Declarações DDL:

```
CREATE GENERATOR <name>;
SET GENERATOR <name> TO <value>;
DROP GENERATOR <name>;
```

Declarações DML no cliente SQL:

```
SELECT GEN_ID(<GeneratorName>, <increment>)
FROM RDB$DATABASE;
<intvar> = GEN_ID(<GeneratorName>, <increment>);
```

Sintaxe recomendada para o Firebird 2

Embora a sintaxe tradicional ainda seja totalmente suportada no Firebird 2, esses são os DDLs equivalentes recomendados:

```
CREATE SEQUENCE <name>;
ALTER SEQUENCE <name> RESTART WITH <value>;
DROP SEQUENCE <name>;
```

E para declarações DML:

```
SELECT NEXT VALUE FOR <SequenceName>
FROM RDB$DATABASE;
<intvar> = NEXT VALUE FOR <SequenceName>;
```

Atualmente, a nova sintaxe não suporta um incremento diferente de 1. Essa limitação será eliminada nas versões futuras. Entretanto, utilize o GEN_ID quando for necessário aplicar outro valor de incremento.

Utilizando declarações para Generators

A disponibilidade de declarações e funções depende de onde elas são utilizadas: cliente SQL (conectado ao servidor) ou PSQL (linguagem utilizada em *Stored Procedures* e *Triggers*).

Criação de um Generator (Insert)

No cliente SQL:

```
CREATE GENERATOR <GeneratorName>;
```

Recomendado para o Firebird 2 e superior:

```
CREATE SEQUENCE <SequenceName>;
```

Em PSQL, não é possível, pois não podemos modificar os metadados do banco de dados em *Stored Procedures* ou *Triggers*, tampouco podemos criar *Generators*.

Obs.: *No Firebird 1.5 e superior, podemos contornar esta limitação com as funcionalidades do EXECUTE STATEMENT.*

Obtendo o valor atual (Select)

No cliente SQL:

```
SELECT GEN_ID(<GeneratorName>, 0) FROM RDB$DATABASE;
```

Essa sintaxe é ainda a única opção para o Firebird 2.

Em PSQL:

```
<intvar> = GEN_ID(<GeneratorName>, 0);
```

No Firebird 2 usamos a mesma sintaxe.

Gerando o próximo valor (Update + Select)

Assim como fazemos para obter o valor atual (GEN_ID), agora utilizando um valor de incremento de 1. O Firebird irá:

1. obter o valor atual do *Generator*;
2. incrementá-lo de 1;
3. retornar o valor incrementado.

No cliente SQL:

```
SELECT GEN_ID(<GeneratorName>, 1) FROM RDB$DATABASE;
```

A nova sintaxe, que é preferida para o Firebird 2, é inteiramente diferente:

```
SELECT NEXT VALUE FOR <SequenceName> FROM RDB$DATABASE;
```

Em PSQL:

```
<intvar> = GEN_ID(<GeneratorName>, 1);
```

Preferida para o Firebird 2 e superior:

```
<intvar> = NEXT VALUE FOR <SequenceName>;
```

Atenção

Na ferramenta isql de linha de comando do Firebird, existem dois comandos adicionais para recuperar o valor atual do Generator:

```
SHOW GENERATOR <GeneratorName>;
SHOW GENERATORS;
```

O primeiro comando recupera o valor atual do Generator especificado, enquanto que o segundo faz o mesmo para todos os Generators no banco de dados que não são do sistema. Os dois comandos equivalentes preferidos para o Firebird são, como poderíamos supor:

```
HOW SEQUENCE <SequenceName>;
SHOW SEQUENCES;
```

Note novamente que esses comandos (SHOW) estão somente disponíveis na ferramenta isql do Firebird. Diferentemente do GEN_ID, não podemos utilizá-los de dentro de outros clientes (a menos que esses clientes sejam front-ends isql).

Configurando um Generator diretamente para um determinado valor (*Update*)

No cliente SQL:

```
SET GENERATOR <GeneratorName> TO <NewValue>;
```

Isso é útil para pré-configurar *Generators* para um valor diferente de 0 (que é o valor padrão após a criação) em, por exemplo, um *script* para criar o banco de dados. Assim como o CREATE GENERATOR, essa é uma declaração DDL (não DML).

Sintaxe preferencial para o Firebird 2 e superior:

```
ALTER SEQUENCE <SequenceName> RESTART WITH <NewValue>;
```

Em PSQL:

```
GEN_ID(<GeneratorName>, <NewValue> - GEN_ID(<GeneratorName>, 0));
```

Obs.: *Isso é mais do que um pequeno truque para fazer o que normalmente não poderíamos e não deveríamos fazer em* ***Stored Procedures*** *e* ***Triggers****: a configuração de* ***Generators****. Esses valores são obtidos e não configurados.*

Eliminando um *Generator* (*Delete*)

No cliente SQL:

```
DROP GENERATOR <GeneratorName>;
```

Preferido para o Firebird 2 e superior:

```
DROP SEQUENCE <SequenceName>;
```

Em PSQL não é possível pois, novamente, não devemos modificar metadados em PSQL.

A eliminação de um *Generator* não libera o espaço por ele ocupado para ser utilizado por um novo *Generator*. Na prática isso raramente é problemático, porque a maior parte dos bancos

de dados não têm as dezenas de milhares de *Generators* que o Firebird permite, portanto existe espaço suficiente para mais.

No entanto, se o banco de dados realmente corre o risco de atingir o teto de 32767, poderemos liberar o espaço dos *Generators* não mais utilizados, executando um ciclo de *backup/restore*. Isso resultará na compactação da tabela RDB$GENERATORS, redefinindo uma uma série contígua de IDs.

Dependendo da situação, o banco de dados restaurado também pode precisar de menos páginas para os valores *Generator*.

Utilizando Generators para criar ID de linhas únicos

Afinal, por que ID de linhas?

A resposta a essa pergunta iria muito além do alcance deste artigo. Se não houver nenhuma necessidade de ter um manipulador genérico e único para cada linha dentro em uma tabela ou não gostarmos da idéia de chaves sem sentido ou representativas em geral, podemos provavelmente pular essa seção.

Um para todos ou um para cada um?

Tudo bem, portanto, desejamos ID de linhas. Uma decisão fundamental e básica a ser tomada, consiste em determinar se utilizaremos um *Generator* único para todas as tabelas ou um *Generator* independente para cada tabela. Isso depende de nós, porém, devemos levar em conta as seguintes considerações.

Na abordagem "um para todas":

- + precisamos somente de um *Generator* único para todos os IDs;
- + teremos um número inteiro que não somente identifica a linha na tabela, porém, no banco de dados inteiro;
- - teremos menos valores de ID possíveis por tabela (isso, na verdade, não deverá ser um problema com *Generators* de 64 bits);
- - logo teremos que lidar com valores de ID grandes mesmo em, por exemplo, busca em tabelas com apenas um número pequeno de registros;
- - provavelmente detectaremos lacunas em uma *Sequence* ID a nível de tabela, pois os valores *Generator* estarão espalhados por todas as tabelas.

Na abordagem "um para cada":

- - teremos que criar um *Generator* para cada tabela com ID no banco de dados;
- - sempre necessitaremos da combinação de ID e nome da tabela para identificar unicamente qualquer linha em qualquer tabela;
- + teremos um contador de inserções simples e robusto por tabela;
- + teremos uma sequência cronológica por tabela: se encontrarmos uma falha na sequência de IDs de uma tabela, esta terá sido causada por um DELETE ou por um INSERT falho.

Poderíamos reutilizar os valores Generator?

Tecnicamente sim, porém não deveríamos. Nunca mesmo! Isso não destruiria apenas a sequência cronológica de Ids (não poderemos mais determinar a idade de uma linha apenas observando o ID), mas quanto mais pensarmos a respeito disso, mais dores de cabeça teremos.

Além disso, é uma absoluta contradição em relação a todo o conceito de identificadores de linha únicos. Portanto, a menos que tenhamos boas razões para reutilizar valores de *Generators* e um mecanismo bem bolado para fazer isso funcionar com segurança em ambientes multi-usuário/multi-transação, não devemos fazer isto.

Generators de IDs ou campos de auto-incremento

Dado um registro recentemente inserido, um ID (no tocante a um número de série único) é facilmente conseguido com os *Generators* e com o *Trigger Before Insert*, como veremos nas subseções seguintes.

Começaremos supondo que temos uma tabela chamada *Test* com uma coluna de ID declarada como inteiro. O nome do *Generator* é GIDTEST.

Trigger Before Insert, versão 1:

```
CREATE TRIGGER TRGTTEST_BI_V1 FOR TESTE
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  NEW.ID = GEN_ID(GIDTEST, 1);
END
```

Problemas com a versão 1 do *Trigger*: funciona muito bem, porém, também desperdiça um valor *Generator* em casos onde já existe um ID fornecido na declaração INSERT. Portanto, será mais eficiente atribuir um valor somente quando não existir nenhum no INSERT.

Trigger Before Insert, versão 2:

```
CREATE TRIGGER TRGTTEST_BI_V2 FOR TESTE
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.ID IS NULL) THEN
  BEGIN
    NEW.ID = GEN_ID(GIDTEST, 1);
  END
END
```

Problemas com a versão 2 do *Trigger*: alguns componentes de acesso têm o mau hábito de auto-preencher todas as colunas em uma declaração INSERT. Aqueles que não foram explicitamente configurados terão designados valores padrão, normalmente 0 para colunas do tipo inteiro.

Nesse caso, a *Trigger* não funcionará: detectará que a coluna ID não tem o estado de NULL, porém, contém o valor 0, portanto não gerará um novo ID. Assim mesmo, poderíamos postar o registro, embora apenas um; o segundo; falhará.

De qualquer maneira, é bom banir o "0" como um valor de ID normal, para evitar qualquer confusão com NULL e 0. Poderíamos, por exemplo, utilizar uma linha especial com um ID 0 para armazenar um registro padrão em cada tabela.

Trigger Before Insert, versão 3:

```
CREATE TRIGGER TRGTTEST_BI_V3 FOR TESTE
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF ((NEW.ID IS NULL) OR (NEW.ID = 0)) THEN
  BEGIN
    NEW.ID = GEN_ID(GIDTEST, 1);
  END
END
```

Bem, agora que temos um *Trigger* de ID robusto e funcional, os parágrafos seguintes explicarão porque geralmente não precisaremos dele em absoluto. O problema básico com IDs atribuídos com *Before Insert* é que esses são gerados do lado servidor após termos enviado a declaração de inserção do lado cliente.

Isso claramente significa que não há nenhuma maneira segura de conhecer do lado cliente, que ID foi gerado para a linha que acabamos de inserir. Poderíamos recuperar o valor do *Generator* do lado cliente depois da inserção, porém, em ambientes multi-usuário, não poderemos, na verdade, ter certeza de que o que obtivemos é o ID da linha desejada (por causa da questão da transação).

No entanto, se obtivermos um novo valor *Generator* antes e postarmos a inserção com esse valor, poderemos simplesmente recuperar a linha com um *Select ... where ID = <genvalue>* para ver quais padrões foram aplicados ou se as colunas foram afetadas pelos *Triggers* de inserção.

Isso funciona especialmente bem, porque normalmente temos uma chave única na coluna de ID e esses são os mais rápidos índices que poderemos obter, são imbatíveis em seletividade e geralmente menores do que índices em colunas do tipo CHAR(n).

Podemos concluir que: devemos criar um *Trigger Before Insert* para tornar absolutamente garantido que cada linha obterá um ID único, mesmo quando nenhum valor de ID for fornecido do lado cliente nas declarações de inserção.

Se tivermos um banco de dados onde o código da nossa aplicação é a única fonte para a inserção de registros, poderíamos então omitir o *Trigger*, porém, nesse caso, deveríamos sempre obter um novo valor *Generator* do banco de dados, antes de emitir a declaração de inserção e incluí-lo nessa.

O mesmo, naturalmente, será válido para inserções através de *Triggers* e *Stored Procedures*.

O que mais podemos fazer com Generators

Vejamos aqui algumas idéias de *Generators* personalizados, além daqueles para a geração de ID de linhas únicos.

Utilizando Generators para atribuir, por exemplo, números únicos para arquivos de transferência

Um uso clássico de *Generators* deve garantir números únicos, sequenciais para, bem, qualquer outra coisa na aplicação além dos ID de linhas tal qual discutido anteriormente. Se tivermos uma aplicação que está transferindo dados para algum outro sistema, poderíamos utilizar *Generators* para identificar com segurança uma única transferência, assinalondo-a com um valor gerado.

Isso ajuda muito no rastreamento de problemas com interfaces entre dois sistemas (e, diferentemente da maior parte dos seguintes, isso realmente funciona com segurança e exatidão).

Generators como contadores de uso para que as Stored Procedures possam prover estatísticas básicas

Imaginemos que acabamos de incorporar uma nova funcionalidade fantástica ao banco de dados com uma *Stored Procedure*. Agora atualizamos os sistemas dos clientes e depois de algum tempo gostaríamos de saber se os usuários na verdade utilizam essa funcionalidade e com que frequência.

Simples: construímos um *Generator* especial que será incrementado somente na *Stored Procedure* e pronto. Apenas uma restrição: não poderemos conhecer a quantidade de transações para as quais houve *rollback* após ou enquanto a *Stored Procedure* executou.

No entanto, pelo menos saberemos com que freqüência os usuários tentaram utilizar a nossa *Stored Procedure*. Poderíamos, além disso, refinar esse método utilizando dois *Generators*: cada um será incrementado logo no início da *Stored Procedure* e o outro precisamente antes do EXIT.

Dessa maneira, poderemos contar quantas tentativas de utilização da *Stored Procedure* foram bem sucedidas: se ambos os

Generators tiverem o mesmo valor, então nenhuma das chamadas à *Stored Procedure* falhou. Naturalmente, ainda não saberemos quantas vezes a(s) transação(ões) que invocam a nossa *Stored Procedure* de fato realizaram o *Commit*.

Generators para simular "Select count(*) from ..."

Existe um problema bem conhecido com o InterBase e com o Firebird, que diz respeito ao fato de que um SELECT COUNT(*) (sem a cláusula *where*) para uma tabela bem grande, pode demorar bastante para executar, pois o servidor deve contar na mão, quantas linhas existem na tabela no momento da solicitação.

Em teoria, poderíamos facilmente resolver esse problema com Generators:

- Criamos um *Generator* especial para contar linhas;
- Construímos um *Trigger Before Insert* que o incrementa;
- Construímos um *Trigger After Delete* que o decrementa.

Isso funciona muito bem e torna desnecessária uma contagem completa de registros: apenas obtêm o valor atual do *Generator*. Realçamos aqui na teoria**,** porque tudo isso vai por água abaixo quando qualquer declaração de inserção falha, porque como foi mencionado antes, os *Generators* estão além do controle da transação. As inserções podem falhar por causa de restrições (violações de chave únicas, campos NOT NULL que são NULL etc.) ou outras restrições de metadados, ou simplesmente porque foi feito o *rollback* da transação que emitiu a inserção.

Poderíamos não ter nenhuma linha na tabela e mesmo assim, o contador de inserções aumentará. Portanto, depende: se conhecermos a porcentagem de inserções que falham (podemos ter uma idéia disso) e somente estamos interessados em uma estimativa da contagem de registros, nesse caso, esse método pode ser útil mesmo que não seja exato.

De vez em quando poderíamos fazer uma contagem de registros "normal" e configurar o *Generator* com o valor exato (re-sincronizar o *Generator*), para que o erro seja minimizado. Existem situações em que os clientes podem conviver com uma informação do tipo "existem *aproximadamente* 2,3 milhões de registros", obtida de forma quase instantânea com um clique do mouse, porém, iria ficar muito insatisfeito se tiver que esperar 10 minutos ou mais para descobrir que existem precisamente 2.313.498.229 linhas...

Listagem 1. Exemplo de uso de Generator para monitorar uma SP

```
CREATE GENERATOR gen_spTestProgress;
CREATE GENERATOR gen_spTestStop;
SET TERM ^;
CREATE PROCEDURE spTest (...)
AS
BEGIN
  (...)
  FOR SCT <LOTS OF DATA TAKING LOTS OF TIME> DO
  BEGIN
    GEN_ID(GEN_SPTESTPROGRESS,1);
    IF (GEN_ID(GEN_SPTESTSTOP,0)>0) THEN EXIT;
    (...NORMAL PROCESSING HERE...)
  END
END^
```

Listagem 2. Usando Generator para monitorar uma SP (com lock)

```
CREATE GENERATOR GEN_SPTESTPROGRESS;
CREATE GENERATOR GEN_SPTESTSTOP;
CREATE GENERATOR GEN_SPTESTLOCKED;
SET TERM ^;
CREATE PROCEDURE SPTEST (...)
AS
DECLARE VARIABLE LOCKCOUNT INTEGER;
BEGIN
  LOCKCOUNT = GEN_ID(GEN_SPTESTLOCKED, 1);
  IF (LOCKCOUNT = 1) THEN
  BEGIN
    (CORPO DA PROCEDURE)
  END
  LOCKCOUNT = GEN_ID(GEN_SPTESTLOCKED, -1);
  WHEN ANY DO
    LOCKCOUNT = GEN_ID(SPTESTLOCKED, -1);
EXIT;
END^
```

Generators para monitorar e/ou controlar Stored Procedures demoradas

Stored Procedures que, por exemplo, geram saídas de relatório para grandes tabelas e/ou para *joins* complexos, podem demorar bastante para executar. Os *Generators* podem ser úteis aqui de duas maneiras: podem fornecer um "contador de progresso", que, por sua vez, poderíamos consultar periodicamente do lado cliente enquanto a *Stored Procedure* está rodando e pode ser utilizado para pará-la (**Listagem 1**).

O código anterior representa apenas um rascunho, porém, dá para termos uma idéia. Do lado cliente, poderíamos emitir um *GEN_ID(gen_spTestProgress, 0)* de modo assíncrono com a extração da linha real (isso é, em uma *thread* diferente), para ver quantas linhas foram processadas e exibir esse valor em algum tipo de janela de progresso.

Poderíamos inclusive construir um *GEN_ID(gen_spTestStop, 1)* para cancelar de fora a *Stored Procedure* a qualquer momento. Embora isso possa ser muito prático, tem uma forte limitação: não é a prova de ambiente multi-usuário. Se a *Stored Procedure* rodar simultaneamente em duas transações, isso irá estragar o *Generator* de progresso, ambas irão incrementar o mesmo contador ao mesmo tempo, portanto, o resultado será inútil.

Além disso, incrementar o *Generator* de parada iria parar imediatamente a *Stored Procedure* em ambas as transações. No entanto, para relatórios mensais, por exemplo, que são gerados por um módulo único, isso pode ser aceitável, como de hábito, depende das nossas necessidades.

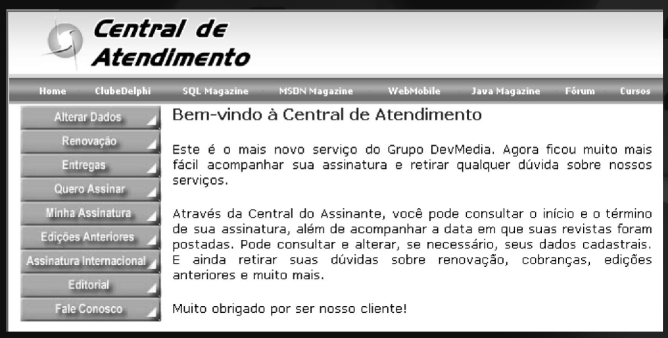
Se quisermos utilizar essa técnica e permitir aos usuários disparar a *Stored Procedure* a qualquer momento, deveremos garantir por algum meio, que a *Stored Procedure* não poderá ser rodada duas vezes. Pensando a respeito disso, tivemos a idéia de utilizar outro *Generator* para isso: chamado de *gen_sp TestLocked* (assumindo, naturalmente, o valor inicial de 0), presente na **Listagem 2**.

A cláusula WHEN ANY trata as exceções, porém, não EXITs normais. Portanto, teríamos que decrementá-lo "na mão". Poderíamos porém, decrementar o *Generator* precisamente antes do EXIT para evitar essa situação. Tomando as devidas precauções, achamos que não existe nenhuma situação em que esse mecanismo poderia falhar.

Conclusão

Vimos neste artigo todos os fundamentos relativos ao uso de Generators no Firebird, além de melhorias presentes na versão 2.0. Mostramos como o banco de dados armazena dados sobre Generators, como utilizá-los em aplicações (client e server) além de usos não-trivias para o uso de Generators. ■

Fale conosco!



Conheça a Central de Atendimento on-line da DevMedia. Agora ficou muito mais fácil de acompanhar sua assinatura, alterar seus dados cadastrais e retirar qualquer dúvida sobre nossos serviços e produtos. Para isso basta ter em mãos seu login e senha.

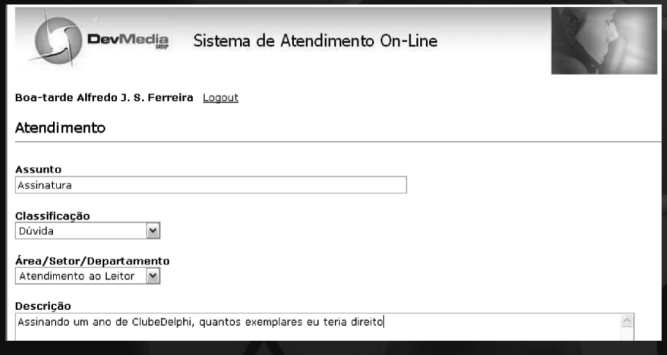
Na Central de Atendimento você:

- Consulta a data de envio de cada exemplar

- Consulta as perguntas mais frequentes

- Renova por boleto ou cartão de crédito

- Consulta e altera seus dados cadastrais



Para entrar em contato com a DevMedia, basta postar sua dúvida na própria Central de Atendimento através de um sistema web, prático, seguro e fácil de usar. Com essa nova ferramenta você tem a garantia de receber suas respostas em menor tempo e com alto padrão de qualidade.

www.devmedia.com.br/central

DevMedia GROUP

Ferramenta nbackup do Firebird 2.0

PAUL VINKENOOG

O *nbackup* é um novo utilitário de backup que vem junto com o Firebird 2.0 e oferece funcionalidades inexistentes no *gbak*, ferramenta de backup pré-existente no Firebird, porém, não substitui esse último. Ambos os programas têm as suas vantagens e desvantagens e irão ainda coexistir provavelmente por algum tempo.

Funcionalidades do nbackup - um resumo

Com o *nbackup*, podemos executar dois grupos diferentes de tarefas:

1. Realizar e restaurar tanto backups completos quanto *incrementais*. Um backup incremental contém somente as alterações ocorridas no banco desde a realização de um backup prévio específico;

2. Bloquear o arquivo do banco de dados principal, o que permite posteriormente realizar o backup, utilizando ferramentas de cópia ou de backup da nossa própria escolha. Nesse caso, o *nbackup* não realiza nenhum backup, apenas cria as condições necessárias para que possamos realizar o backup com segurança. Aqui também existe uma previsão para restauração.

Ambos os modos podem funcionar em um banco de dados ativo, sem interferir com os usuários conectados. O backup criado sempre refletirá o estado do banco de dados no início da operação. Nesse sentido, o *nbackup* não tem diferença com relação ao *gbak*.

Vantagens do nbackup

• *Ambos os modos:* alta velocidade (tão alta quanto o hardware e OS permitam), pois o *nbackup* não enxerga os dados reais. No modo de backup o conteúdo é escrito de forma mais ou menos "cega" no arquivo backup;

• *Modo backup/restore:* economia de tempo e de espaço em disco, pois não precisamos fazer toda vez um backup completo. Essa modalidade pode fazer uma enorme diferença com ban-

cos de dados com tamanho de gigabytes;

• *Modo bloqueio/desbloqueio (lock/unlock):* liberdade total de escolha das ferramentas de backup, de cópia e/ou de compressão.

Limitações do nbackup

• O *nbackup* não efetuará uma varredura e compactará o banco de dados da maneira como o *gbak* o faz;

• Não poderemos alterar o dono do banco de dados dentro de um ciclo *nbackup* de backup/restauração, tal como faríamos com o *gbak*;

• O *nbackup* não pode construir backups transportáveis (*transportable backups*), isso é: backups passíveis de serem restaurados em uma plataforma incompatível ou sob outra versão de servidor;

• No momento, o *nbackup* não pode ser utilizado em bancos de dados multi-arquivo;

• O *nbackup* consegue suportar apenas bancos de dados locais.

Nas próximas seções, discutiremos extensivamente as várias funcionalidades do *nbackup*.

Realizando e restaurando backups

Em primeiro lugar: o *nbackup.exe* encontra-se no diretório *bin*, na pasta de instalação do Firebird. Assim como na maioria das ferramentas que vem com o Firebird, o *nbackup* não possui interface gráfica, para executá-lo, devemos utilizar a linha de comando (ou a partir de um arquivo lote, *batch file*, ou de uma aplicação).

Realizando backups completos

Para realizar um backup completo do banco de dados, a sintaxe de comando é:

```
nbackup [-U <usuário>-P <senha>]-B 0
  <banco de dados> [<backupfile>]
```

Por exemplo:

```
C:\Databases> nbackup -B 0 inventory.fdb
  inventory_1-Mar-2006.nbk
```

Comentários:

• O parâmetro *-B* significa backup (nossa!). *O nível de backup* 0, indica um backup completo. Backups de níveis maiores do que 0 são utilizados para backups incrementais, discutiremos mais tarde;

• Em lugar de um nome de arquivo do banco de dados, podemos também especificar um alias;

• Em lugar de um nome de arquivo backup, podemos também especificar *stdout*. A especificação dessa opção, direcionará o backup para a saída padrão, de onde poderá ser redirecionada para, por exemplo, um dispositivo de armazenamento em fita ou uma ferramenta de compressão;

• Os parâmetros *-U* (usuário) e *-P* (senha) podem ser omitidos nos seguintes casos:

  - Quando fizermos o login como super-usuário (root, administrador.) ou
  - Quando as variáveis de ambiente ISC_USER e ISC_PASSWORD estiverem configuradas.

Para maior clareza e brevidade, esses parâmetros não serão utilizados nos exemplos;

• Os parâmetros diferentes (*-B*, *-U* e *-P*) poderão ocorrer em qualquer ordem. Naturalmente, cada parâmetro deve ser imediatamente seguido pelo(s) seu(s) próprio(s) argumento(s). No caso de *–B* existem três deles: nível de backup, banco de dados e arquivo backup, nessa ordem;

• Caso o parâmetro *-B* for especificado por último, poderemos omitir o nome do arquivo backup. Nesse caso, o *nbackup* gerará um nome de arquivo baseado no nome do banco de dados, nível de backup e a data e hora atuais. Isso poderá resultar em um conflito de nomes (e uma falha do backup), caso dois comandos de backup do mesmo nível, forem emitidos simultaneamente.

> **Importante:** *Não utilizar o **nbackup** com bancos de dados multi-arquivo, pois poderá resultar em corrupção e perda de dados, apesar do fato de que o **nbackup** não alertará a respeito de tal comando.*

Algumas observações com relação ao funcionamento interno

O que vem a seguir não representa o conhecimento mínimo necessário para a utilização do *nbackup*, apenas fornece uma idéia básica do que acontece por trás dos panos durante a execução do *nbackup -B*:

1. Em primeiro lugar, o arquivo do banco de dados principal é bloqueado mediante a alteração de um *flag* de estado interno. A partir desse momento, qualquer mudança no banco de dados será escrita em um arquivo temporário, o arquivo diferencial ou arquivo delta;

2. A seguir, o backup real é construído. Esse backup não é uma cópia de arquivo que possa ser utilizada diretamente, a restauração deve ser realizada também pelo *nbackup*;

3. Assim que o backup tenha sido completado, o conteúdo do arquivo delta estará integrado ao arquivo de banco de dados principal. Nesse momento, o banco de dados é desbloqueado (o *flag* volta para normal) e o delta é removido.

As funcionalidades dos passos 1 e 3 são fornecidas por duas novas cláusulas SQL: ALTER DATABASE BEGIN BACKUP e ALTER DATABASE END BACKUP. Ao contrário do que os nomes sugerem essas cláusulas não realizam o backup real, na verdade, apenas criam as condições para que o backup do arquivo do banco de dados principal possa ser realizado com segurança.

Ainda mais: não precisamos mesmo emitir esses comandos; o *nbackup* o fará automaticamente, no momento apropriado.

Restauração de um backup completo

Um backup completo é restaurado da seguinte maneira:

```
nbackup [-U <usuário>-P <senha>]-R  <banco de dados> [<backupfile>]
```

Por exemplo:

```
C:\Databases> nbackup -R inventory.fdb inventory_1-Mar-2006.nbk
```

Comentários:

• Não especificamos um nível para a restauração;

• Na restauração, o parâmetro *-R* **deve** vir último, por motivos que se tornarão claros posteriormente;

• Se o banco de dados especificado já existir e não houver nenhuma conexão, o mesmo será sobrescrito sem qualquer mensagem de alerta. Se houver usuários conectados, a restauração falhará e receberemos uma mensagem de erro;

• Note também, que poderemos omitir o nome do arquivo de backup. Caso o omitirmos, o *nbackup* o solicitará.

Realizando backups incrementais

Para realizarmos um backup incremental (diferencial), especificamos um nível de backup maior do que 0. Um backup incremental de nível *N*, sempre contém as modificações do banco de dados desde o nível de backup mais recente *N -1*. Exemplos:

Um dia depois do backup completo (nível 0), realizamos outro de nível 1:

```
C:\Databases> nbackup -B 1 inventory.fdb
  inventory_2-Mar-2006.nbk
```

Esse backup só conterá as modificações do dia anterior. No dia seguinte, fazemos novamente um outro de nível 1:

```
C:\Databases> nbackup -B 1 inventory.fdb
  inventory_3-Mar-2006.nbk
```

Esse backup conterá as modificações dos dois dias anteriores desde o backup completo e não somente as do backup de nível 1 prévio. Duas horas depois, realizamos um backup de nível 2:

```
C:\Databases> nbackup -B 2 inventory.fdb
  inventory_3-Mar-2006_2.nbk
```

Esse backup mais recente conterá somente as modificações desde o backup de nível 1 mais recente, isso é: das últimas duas horas.

> **Obs.:** *Todos os comentários a respeito dos backups completos se aplicam também aos backups incrementais. Novamente: não utilize o* ***nbackup*** *com bancos de dados multi-arquivo.*

Restauração de backups incrementais

Para restaurarmos backups incrementais, devemos especificar a cadeia completa de arquivos de backup, desde o nível 0, até aquele que se deseja restaurar. O banco de dados será sempre restaurado desde o começo, passo a passo (é essa adição gradual até que o banco de dados esteja restaurado, que deu a origem à denominação backup incremental).

A sintaxe formal é:

```
nbackup [-U <user> -P <password>] -R
  <database> [<backup0> [<backup1> [...]]]
```

Portanto, para restaurar o backup de nível 2 do exemplo anterior, procederemos da seguinte maneira:

```
C:\Databases> nbackup -R inventory.fdb
  inventory_1-Mar-2007.nbk
  inventory_3-Mar-2007.nbk
  inventory_3-Mar-2007_2.nbk
```

Naturalmente, a linha foi dividida aqui apenas por razões de *layout*, na verdade escrevemos o comando completo e só pressionamos ENTER ao final do mesmo. Comentários (adicionais aos

comentários relativos à restauração de um backup completo):

• Como não sabemos antecipadamente quantos nomes de arquivo virão após a chave *-R* (já que não especificamos um nível ao restaurar), o *nbackup* considera que todos os argumentos após o *-R* são nomes de arquivos de backup. Esse é o motivo porque nenhum outro parâmetro (*-U* ou *-P*) pode vir após a lista de nomes de arquivo;

• Não existe nenhum limite formal quanto ao número de níveis de backup, porém, na prática, raramente fará sentido ultrapassar os níveis 3 ou 4.

Links não encadeados

O que aconteceria se acidentalmente omitirmos um arquivo ou especificarmos uma série de arquivos que não pertence a um conjunto específico? Imaginemos que acidentalmente especifiquemos *inventory_2-Mar-2007.nbk* em vez de *inventory_3-Mar-2007.nbk* no exemplo anterior.

Ambos são arquivos de backup de nível 1, portanto, em ambos os casos, obteremos séries corretas de nível "0, 1, 2". No entanto, o nosso arquivo de nível 2 é incremental com relação ao backup de nível 1 de 3 de Março e não para o do dia 2 de Março.

Tal erro nunca nos conduzirá a um banco de dados incorretamente restaurado. Cada arquivo de backup tem o seu ID único. Além disso, cada arquivo backup de nível 1 ou superior, contém o ID do backup que foi baseado.

Ao acionar a restauração, o *nbackup* verifica esses IDs e caso a cadeia esteja interrompida em algum lugar, a operação é abortada e recebemos uma mensagem de erro.

Uma aplicação prática

Um esquema de backup incremental baseado em *nbackup* poderia ter os seguintes passos:

• Cada mês, um backup completo (de nível 0) é construído;

• Cada semana, um backup de nível 1;

• Diariamente, um backup de nível 2;

• De hora em hora, um backup de nível 3.

Caso todos os backups tenham sido conservados, poderemos restaurar o banco de dados para o seu estado original, em qualquer hora do passado. Para cada ação de restauração, um máximo de quatro arquivos de backup será utilizado.

Naturalmente, planejaremos de tal maneira que os backups mais grandes, que consomem muito tempo, sejam construídos em períodos fora das horas de pico. Nesse caso, os backups de níveis 0 e 1, podem ser construídos nos fins de semana e os de nível 2 à noite.

Se não quisermos guardar os backups para sempre, poderemos acrescentar um horário para sua eliminação:

• Backups de Nível 3 são eliminados depois de 8 dias;

• Backups de Nível 2 depois de um mês;

• Backups de Nível 1 depois de seis meses;

• Backups completos depois de dois anos, porém, o primeiro de cada ano deverá ser guardado.

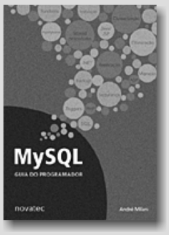
Naturalmente, isso é apenas um exemplo. O procedimento mais adequado a ser adotado, dependerá da aplicação, do tamanho do banco de dados, da sua atividade etc.

Vamos em frente?

Nesse ponto, já sabemos tudo o que é necessário para realizar e restaurar backups completos e/ou incrementais com o *nbackup*. Caso quisermos utilizar ferramentas da nossa própria escolha para realizar o backup do banco de dados Firebird, devemos continuar a leitura. Se esse não for o caso, então boa sorte ao trabalhar com o *nbackup*!

Desconto Especial!

Lançamento



MySQL
Guia do Programador
Novatec
R$ 74,00

UML Essencial
3ª ed.
Bookman
R$ 61,00

Dominando Netbeans
Construa aplicativos Java tanto em Desktop, como para Web
Ciência Moderna
R$ 76,00

Dominando Firebird
Uma referência para Desenvolvedores de Bancos de Dados
Ciência Moderna
R$ 179,00

MySQL 5 Interativo
Guia Básico de Orientação e Desenvolvimento
Érica
R$ 89,00

COMPRE JÁ e garanta um DESCONTO ESPECIAL!
Utilize o código de cupom LDP02 quando solicitado.*
* Válido até 30/04/2007

Frete Grátis
Para todo o Brasil!

Até 6x sem juros no Visa, Mastercard e Hipercard.*
*Parcela mínima de R$ 20,00

Outras facilidades de pagamento:

Tel: (11) 4062-5152

livrosdeprogramacao.com.br

Bloqueio e desbloqueio

Caso preferirmos utilizar ferramentas de backup da nossa própria escolha ou apenas realizar uma simples cópia de arquivo, devemos utilizar o modo de bloqueio/desbloqueio do *nbackup's*. Bloqueio aqui quer dizer que o arquivo do banco de dados principal é congelado temporariamente, isso não quer dizer que não poderá haver nenhuma modificação no banco de dados.

Tal como no modo de backup, as modificações são dirigidas para um arquivo delta temporário e após o desbloqueio, o arquivo delta será fundido com o arquivo principal. Não existe nenhuma interface gráfica; o *nbackup* deve ser acionado a partir da linha de comando ou a partir de um arquivo de lote ou de uma aplicação.

Bloqueando o banco de dados e realizando o backup

Uma sessão típica onde faríamos o backup, teria a seguinte sequência de passos:

1. Bloqueamos o banco de dados com a chave *-L* (lock - bloqueio):

```
nbackup [-U <user> -P <password>] -L <database>
```

2. Agora copiamos/realizamos o backup/efetuamos a compactação (ZIP) do arquivo do banco de dados, escolhendo uma ferramenta (uma simples cópia de arquivo também é possível).

3. Desbloqueamos o banco de dados com a chave *-N* (desbloqueio):

```
nbackup [-U <user> -P <password>] -N <database>
```

O último comando fará também com que qualquer modificação que tenha sido gravada no arquivo delta, seja fundida com o arquivo principal. O backup assim construído, contém os dados do modo como estavam no momento em que o banco de dados foi bloqueado, independentemente de quanto tempo esse estado de bloqueio durou e independentemente de quanto tempo aguardamos antes de realizar o backup real.

> **Obs.:** *Tudo o que foi dito com relação à realização de backup/restore, também se aplica às chaves de bloqueio/desbloqueio: não devemos utilizá-las com bancos de dados multi-arquivo. Enquanto não houver mudanças nesse sentido, não devemos deixar em absoluto o* ***nbackup*** *solto em bancos de dados multi-arquivo.*

Restauração de um backup construído após o "nbackup -L"

Uma cópia de um banco de dados bloqueado é um banco de dados bloqueado também, portanto, não podemos apenas copiá-lo e começar a utilizá-lo. Caso o banco de dados original for perdido ou for danificado e precisarmos restaurar a cópia construída (ou caso desejarmos instalar a cópia em outra máquina), devemos proceder como a seguir:

1. Copiar/restaurar/abrir o arquivo ZIP do banco de dados realizado com as correspondentes ferramentas.

2. Agora, desbloquear o banco de dados, não com a chave *-N*, porém, com a chave *-F* (*fixup*):

```
nbackup -F <database>
```

Com que finalidade existem duas chave de desbloqueio *-N* e *-F*?

- *-N* se certifica que quaisquer modificações realizadas desde o bloqueio mediante a chave *–L*, sejam fundidas com o arquivo do banco de dados principal;
- Depois que o banco de dados retorna ao modo leitura/escrita normal e o arquivo temporário tenha sido eliminado;
- *-F* apenas altera o *flag* de estado do banco de dados autorestaurado para normal;

Portanto, utilizamos:

- *-N* depois de termos construído uma cópia/backup (para reverter o *-L* anteriormente acionado);
- *-F* depois de ter restaurado o backup.

> **Obs.:** *ter chamado a chave* **-F** *de* **Fixup** *(consertar), não foi uma escolha das mais felizes. No fim das contas, não conserta nada, somente desbloqueia o banco de dados. O* **flag -N** *(desbloquear), por outro lado, não só realiza o desbloqueio, como também um conserto (***fixup***). No entanto, teremos que conviver com isso.*

Por baixo dos panos

Essa seção não acrescenta nenhum conhecimento indispensável, porém, fornece algumas informações extras para nos aprofundarmos na compreensão das várias chaves. O *nbackup -L* faz o seguinte:

Conecta ao banco de dados, inicia uma transação, chama ALTER DATABASE BEGIN BACKUP, efetua o *commit* da transação e desconecta do banco de dados. O *nbackup -N* segue os mesmos passos, porém, utilizando END BACKUP. O *nbackup -F* funciona da seguinte maneira:

O arquivo do banco de dados restaurado é aberto, dentro do arquivo, o *flag* de estado é modificado de bloqueado (*nbak_state_stalled*) para normal (*nbak_state_normal*) e após o arquivo é fechado novamente.

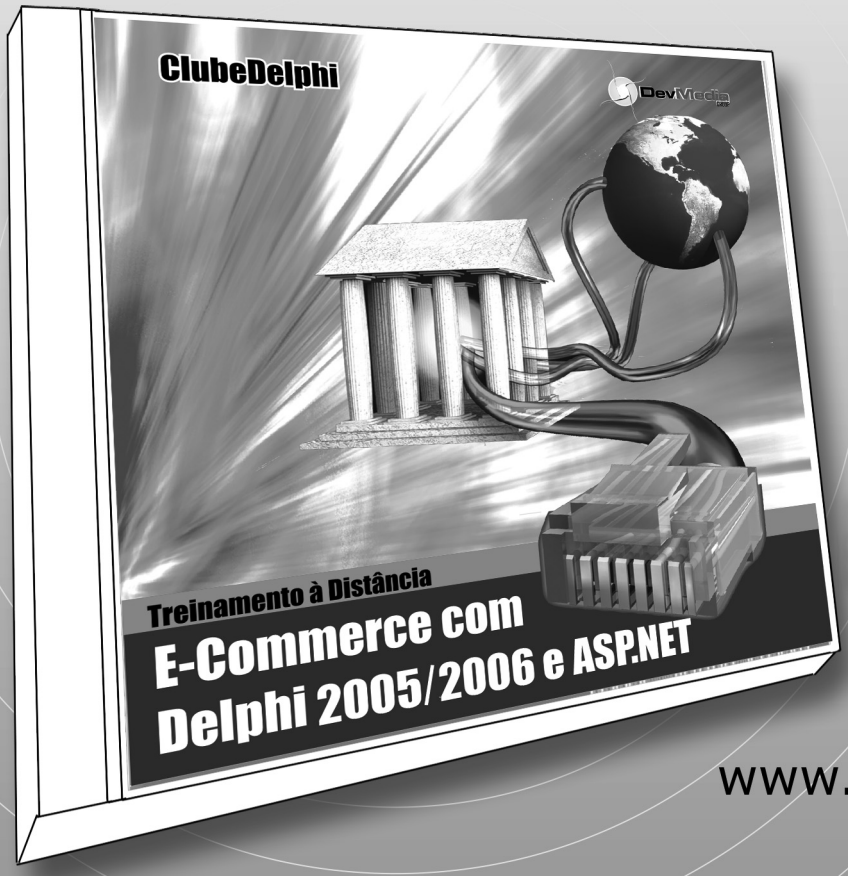
> **Obs.:** *O nbackup -F funciona em um nível de arquivo puro e pode, portanto, ser executado sem o servidor Firebird rodando. Quaisquer parâmetros -U ou -P acrescentados ao comando serão completamente ignorados.*

Conclusão

Vimos neste artigo como funciona o utilitário nbackup para criação de backups, inclusive incrementais, no Firebird 2.0. Vimos as várias opções disponíveis na linha de comando, além de detalhes internos de funcionamento. ■

Treinamentos a distância.

Aprenda de forma prática, eficiente e econômica.



E-Commerce com Delphi 2005/2006 e ASP.NET

Aprenda a desenvolver poderosas aplicações Web através do Delphi 2005/2006 e as tecnologias ASP.NET, WebBroker, Intraweb e WebSnap.
O curso conta com módulos textuais;
35 horas de vídeo-aulas;
fórum exclusivo para os alunos e suporte on-line.

COMPRE JÁ O SEU!

www.devmedia.com.br/curso

Outros Cursos

Delphi e Relatórios

Com este curso você desenvolverá relatórios através do Delphi 7, do Delphi 2005 e Delphi 2006, utilizando as ferramentas Rave Reports, Quick Reports e Crystal Report. Além do módulo textual o curso contém mais de 20 horas de vídeo-aulas; fórum exclusivo e suporte

Evento à distância - Introdução ao .NET

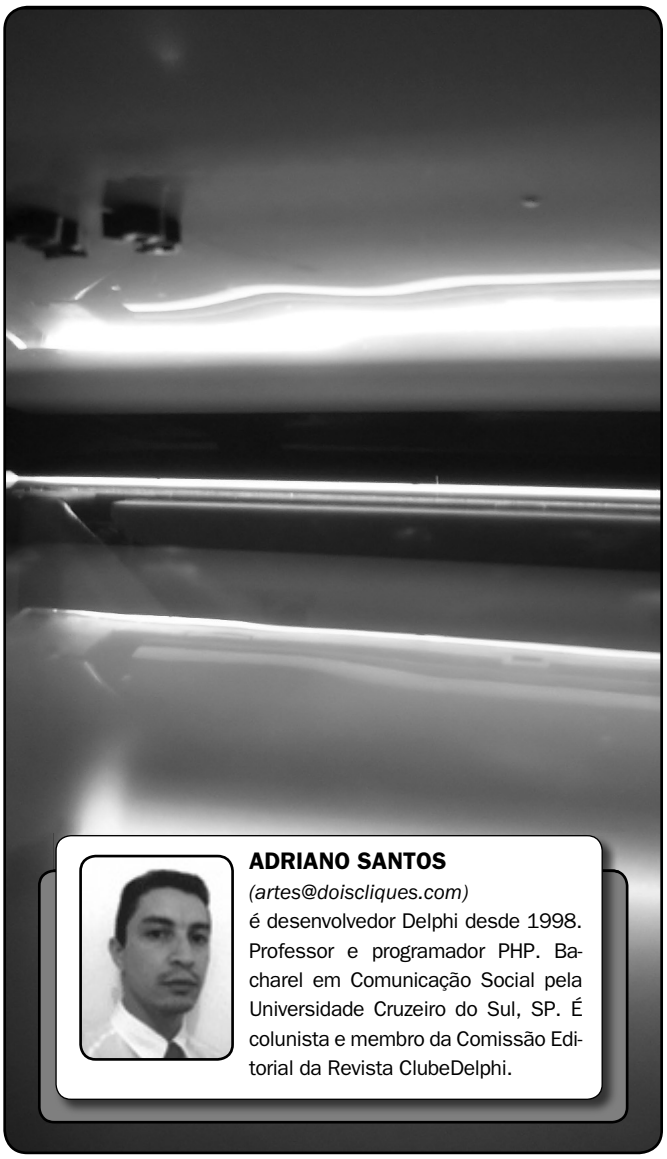
Conheça os recursos do ASP.NET com o Visual Studio 2003 e aprenda a criar páginas ASP.NET; Webservices; trabalhar com a IDE do VS; componentização; DataGrid e criar aplicações de BD com o ADO.NET! São 6h de palestras em formato de vídeo-aulas e você pode tirar suas dúvidas com os palestrantes!

Evento à distância - Novidades do Delphi 2005

Conheça os recursos do Delphi 2005 e aprenda a criar páginas ASP.NET; Webservices; trabalhar com a nova IDE do Delphi; utilizar programação OO no Delphi 2005; trabalhar em equipe com o novo Starteam e criar aplicações de BD com o ADO.NET!

DevMedia

Como escanear e armazenar documentos com Delphi



ADRIANO SANTOS
(artes@doiscliques.com)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É colunista e membro da Comissão Editorial da Revista ClubeDelphi.

Certamente uma das maiores vantagens em informatizar uma empresa hoje, resume-se em armazenar em bancos de dados milhares de informações geradas no dia-a-dia, tais como dados de vendas, cotações, relatórios de movimentação de mercadorias, extratos bancários e uma série de outros relatórios importantes para o bom andamento dos negócios.

Porém, o que vemos é um grande acumulo de papéis, muitos deles gerados pelo próprio sistema da empresa, já que fica muito mais fácil visualizá-los impressos. Pensando em diminuir a quantidade de papel, algumas empresas costumam digitalizar seus documentos e armazená-los no computador para posterior análise ou apenas no intuito de arquivá-los.

Neste artigo veremos como desenvolver um cadastro de documentos e usufruir de um recurso pouco utilizado no dia-a-dia: o escaneamento de imagens. Vamos colocar a mão na massa.

> **Nota:** Para este artigo é necessário basicamente um scanner de mesa devidamente instalado, tanto fisicamente quanto logicamente, através do software que acompanha o aparelho.

Entendendo o projeto

Nosso projeto terá como objetivo interagir com o software do scanner instalado no Windows. Faremos com que o projeto invoque o aplicativo e aguarde até que o usuário conclua a digitalização da imagem.

Logo em seguida a imagem é transferida para um *DBImage* que é salvo automaticamente em nossa base de dados. Para isso, usaremos um componente *freeware* pouco conhecido que faz todo o trabalho de captura e interação com o usuário. Ele tem a função de "conversar" com a biblioteca TWAIN_32.DLL presente no sistema operacional.

Instalando o componente

Junto com o exemplo deste artigo segue o pacote (DPK) de instalação do *AcquireImage* (*TAcquireImage.dpk*). Basta abrir o Delphi, clicar em *Open*, localizar o arquivo e em seguida compilar e instalar (**Figura 1**).

> **Nota:** O **AcquireImage** foi testado nas versões 6, 7 e 2006 do Delphi.

Não esqueça de adicionar o caminho dos arquivos fonte ou DCU no *Library Path* do Delphi, no menu *Tools>Enviroment Options>Library>Library Path*.

Criando o banco de dados

Em seguida criaremos a base de dados que fará parte do nosso exemplo. A estrutura do banco (tabela DOCUMENTOS) pode ser vista na **Tabela 1**. O *script* de criação do banco encontra-se na **Listagem 1**

Listagem 1. Criação do banco de dados e tabela

```
SET SQL DIALECT 3;
SET NAMES WIN1252;
CREATE DATABASE <caminho>\DOCUMENTOS.FDB'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 4096
DEFAULT CHARACTER SET WIN1252;

CREATE TABLE DOCUMENTOS (
  NOME_ARQUIVO  VARCHAR(100) NOT NULL,
  DESCRICAO     VARCHAR(50) NOT NULL,
  DOCUMENTO     BLOB SUB_TYPE 0 SEGMENT SIZE 80
);

ALTER TABLE DOCUMENTOS
ADD CONSTRAINT PK_DOCUMENTOS
PRIMARY KEY (NOME_ARQUIVO);
```

> **Nota:** Veja que o nome do arquivo (NOME_ARQUIVO) é a chave primária da tabela.

Desenvolvendo o projeto e acesso a dados

Vamos "desenhar" uma tela simples de cadastro com os típicos botões *Incluir, Alterar, Gravar, Cancelar* e *Excluir*. Inclua também no layout um botão *Escanear Documento* e outro botão para *Sair*. Para visualizar os dados basta incluir um conjunto de componentes *Data-aware* como segue na **Tabela 2**.

Campo	Tipo/Tamanho
NOME_ARQUIVO	Varchar(100) Not Null
DESCRICAO	Varchar(50) Not Null
DOCUMENTO	BLOB SUB_TYPE 0

Tabela 1. Tabela DOCUMENTOS

E para o acesso ao banco de dados vamos usar, como costumo chamar, o quarteto fantástico do dbExpress. Insira no formulário um *SqlConnection* e um *SqlDataSet* da paleta *dbExpress* e dê os nomes de "sqlConexao" e "sdsDocumentos", respectivamente.

No *SqlConnection*, clique duas vezes e crie uma nova conexão apontando para o arquivo *Documentos.fdb* criado anteriormente. Em seguida selecione o *SqlDataSet* e em sua propriedade *SqlConnection* aponte-o para para *sqlConexao*.

Adicione os componentes *DataSetProvider* ("dspDocumentos"), *ClientDataSet* ("cdsDocumentos") e *DataSource* ("dtsDocumentos"), todos da paleta *Data Access*. Configure-os de acordo com a **Listagem 2**.

Listagem 2. Configurando o acesso ao banco de dados

```
SdsDocumentos
CommandText:
  "SELECT * FROM DOCUMENTOS ORDER BY NOME_ARQUIVO"
SqlConnection: sqlConexao
dspDocumentos
DataSet: sdsDocumentos
cdsDocumentos
ProviderName: dpsDocumentos
dtsDocumentos
DataSet: cdsDocumentos
```

Para concluir, adicione os botões necessários para as operações citadas anteriormente: *Incluir, Alterar, Gravar, Cancelar, Excluir, Escanear Documento* e *Sair*.

Insira também um *RadioGroup* ("rdgTipoImagem"). Faremos uso dele para escanear a imagem em formato *Bitmap*, *Jpeg* ou direto para a memória. Por isso, em sua propriedade *Items* adicione as seguintes *strings*: "Bitmap", "JPEG" e "Clipboard (Memória)".

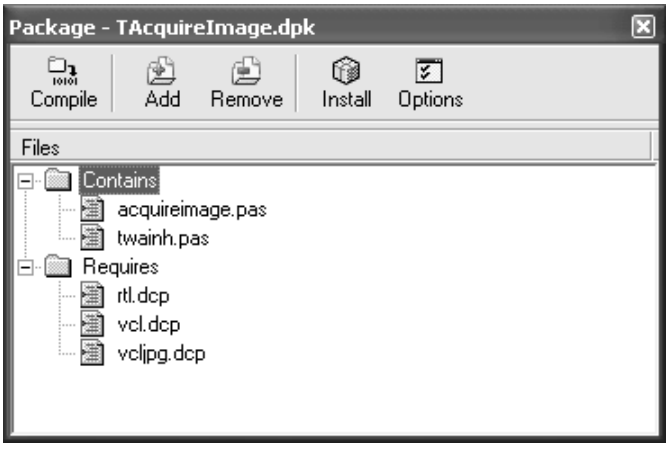


Figura 1. Instalando o AcquireImage

Componente	Nome	Propriedade = Valor
DBGrid	dbgDocumentos	DataSource = dtsDocumentos
DBEdit	dbeDescricao	DataSource = dtsDocumentos; FieldName = DESCRICAO
DBEdit	dbeNomeArquivo	DataSource = dtsDocumentos; FieldName = NOME_ARQUIVO
DBImage	dbImagem	DataSource = dtsDocumentos; FieldName = DOCUMENTO

Tabela 2. Componentes visuais para a tabela DOCUMENTOS

Ainda se tratando de componente, falta inserir o responsável por interagir com o scanner. Clique na paleta *Others* e insira um *AcquireImage* no formulário. Nesse ponto do artigo nossa tela deverá se parecer com a **Figura 2**.

Não entraremos em detalhes quanto ao desenvolvimento do formulário principal, pois basta incluir rotinas simples para Inclusão, Alteração, Exclusão, Gravação etc. Vamos focar no *AcquireImage*.

Listagem 3. Código do btnCapturar

```
procedure TfrmPrincipal.btnCapturaClick(Sender: TObject);

var
  Jpg: TJpegImage;
  Bmp: TBitmap;
  Flag: Boolean;
  S: string;

begin
  Flag := False;

  with aiScanearImagem do
  begin
    if LoadTWainModule then
    begin
      try
        OpenSourceManager;
        S := GetSource(false);
        SelectSource(S);
        OpenSource;

        case rdgTipoImagem.ItemIndex of
          0: AcquireBmp(Bmp);
          1: AcquireJpg(Jpg, 50);
          2: AcquireToClipboard;
        end;

        Flag := True;
      finally
        CloseTWainSession;
        UnloadTWainModule;
      end;
    end

    else
      MessageDlg(
        'Erro ao carregar a biblioteca TWAIN_32.DLL',
        mtError, [mbOk], 0);
  end;

  if Flag then
    case rdgTipoImagem.ItemIndex of
      0: dbImagem.Picture.Bitmap := Bmp;
      1: dbImagem.Picture.Bitmap.Assign(Jpg);
      2: dbImagem.Picture.Bitmap.
           LoadFromClipboardFormat(cf_BitMap,
           ClipBoard.GetAsHandle(cf_Bitmap), 0);
    end;

end;
```

Entendendo o componente

O *AcquireImage* funciona basicamente como uma ponte para a biblioteca TWAIN_32.dll que faz todo o trabalho de escanear e guardar a imagem em memória através do aplicativo do scanner. Por isso, internamente o mesmo faz referência à DLL e a carrega para que possamos usá-la.

Quando solicitado, o componente invoca o programa do scanner e aguarda até que o usuário cancele ou solicite a digitalização. Dessa forma, seu resultado é retornado ao *AcquireImage* que por sua vez o armazena.

Capturando a imagem do documento

O que faremos é usar a imagem capturada para atualizar o campo DOCUMENTO do tipo BLOB no banco de dados e salvar, fazendo com que a figura seja salva para futuras consultas. Clique duas vezes no *btnCapturar* e digite o código da **Listagem 3**.

Veremos agora o detalhamento do código de captura começando pelo método *Lo-*

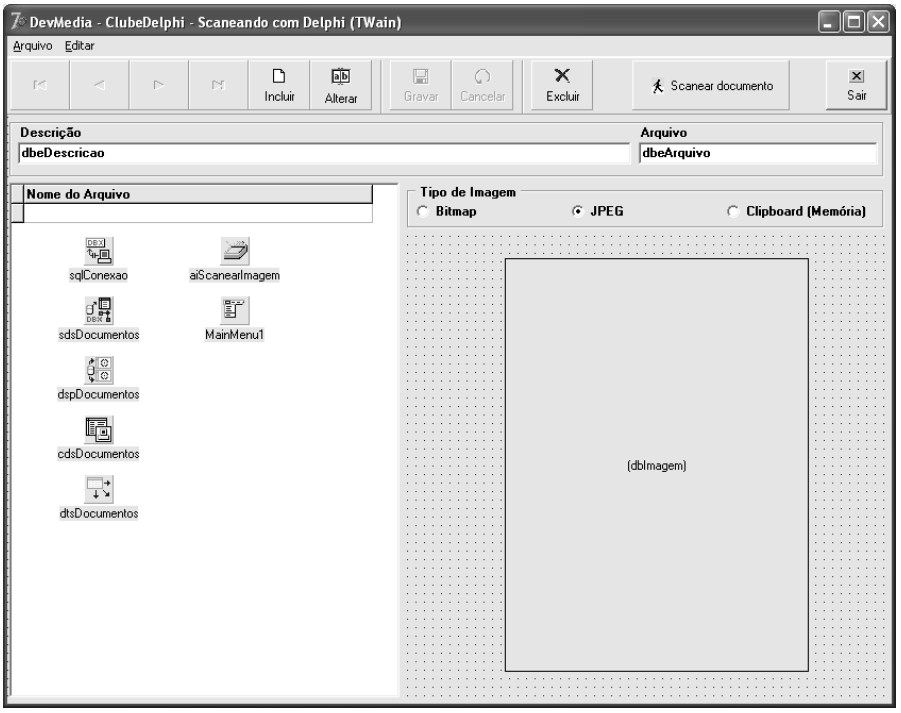


Figura 2. Formulário principal da aplicação

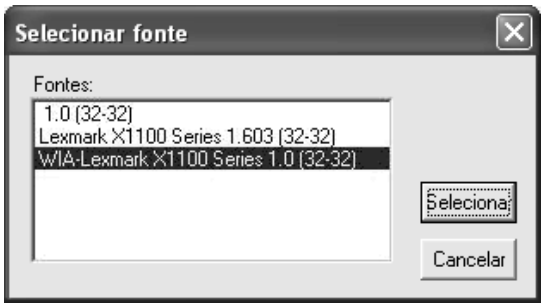


Figura 3. Gerenciador de dispositivo (fonte) a ser utilizado

Figura 4. Gerenciador de digitalização do scanner

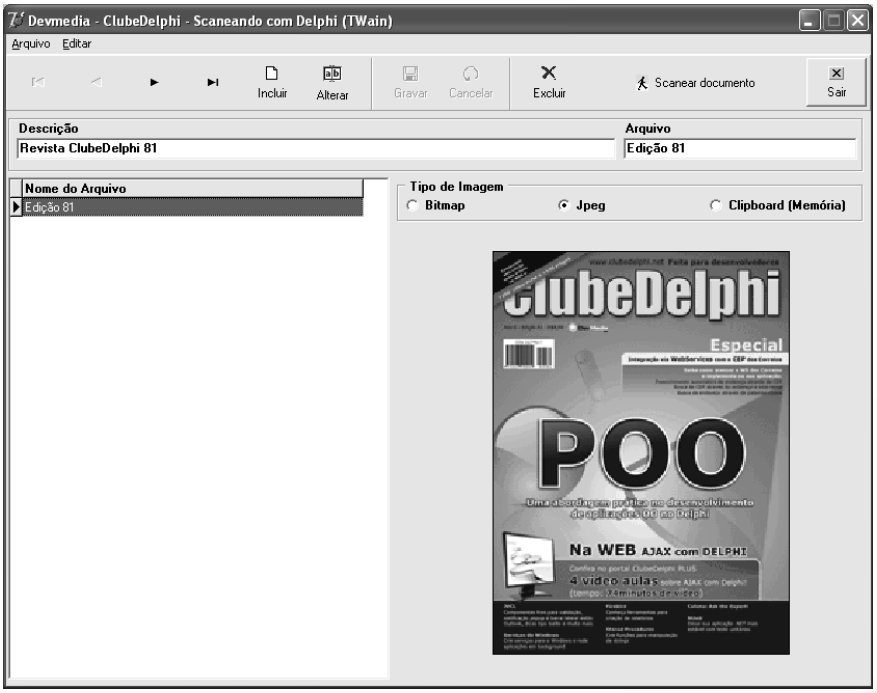


Figura 5. Imagem escaneada e salva no banco de dados do aplicativo

adTWainModule que retorna *True* se conseguiu carregar e *False* se ocorreu algum erro durante o processo. Os erros mais prováveis ocorrem quando a DLL já está em uso por outro processo ou mesmo se ela não existir. Caso ocorra algum problema enviamos uma mensagem ao usuário.

Logo em seguida abrimos o *SourceManager* usando o *OpenSourceManager*. O método se encarrega de verificar quais são as "fontes" que estão disponíveis no sistema operacional. Essas fontes são os aplicativos de digitalização que podem ser utilizados (**Figura 3**).

Logo em seguida chamamos a função *GetSource*, que abre a janela de dispositivos e aguarda até que o usuário escolha um na lista. Feito isso invocamos a função *OpenSource* para preparar a abertura do aplicativo, e em seguida definimos para onde será enviada a imagem capturada: *Bitmap*, *Jpeg* ou Clipboard (memória).

Nesta última fase, usamos o *RadioGroup* para definir e chamar a função necessária, de acordo com o tipo de imagem desejada. Conforme a seleção, chamamos: *AcquireBmp(Bmp)* para adquirir em formato *Bitmap*, *AcquireJpg(Jpg, 50)* para *Jpeg* ou *AcquireToClipboard* para enviar a imagem a memória. São essas três funções responsáveis por chamar efetivamente o gerenciador de digitalização do scanner (**Figura 4**).

Por fim marcamos a variável *Flag* para *True* pra sinalizar que a digitalização correu bem. Não podemos esquecer de fechar a sessão do módulo DLL (*CloseTWainSession*) e descarregar a mesma da memória (*UnloadTWainModule*).

O último passo é verificar se a *Flag* é verdadeira, indicando que a digitalização aconteceu, e atualizar o *DBImage*. Na **Figura 5** podemos ver a imagem escaneada e salva no banco de dados.

Conclusão

O armazenamento de documentos no sistema pode ser uma grande vantagem para quem quer economizar espaço no escritório e sumir com boa parte do papel que é gerado na empresa. Neste artigo, aprendemos a desenvolver uma aplicação capaz de digitalizar e armazenar documentos escaneados. ■

Envio automatizado de emails com Delphi e Outlook - Parte 2

MURILO COSTA M. FILHO
(murilo.filho@asa-sistemas.com)
é empresário e desenvolvedor de aplicações Delphi desde 2000. É Microsoft Certified Professional (MCP) e desenvolve softwares corporativos personalizados pela ASA - Assessoria em Sistemas Avançados (www.asa-sistemas.com).

Continuando a série de artigos sobre automação Outlook via Delphi, dessa vez demonstrarei alguns recursos mais avançados, como anexar arquivos e formatar mensagens em HTML.

Na edição 82, vimos como percorrer a tabela de funcionários do *Northwind* e, para cada funcionário cadastrado, gerar um e-mail com dados de uma mala direta, substituindo as variáveis pelos dados da tabela. Agora aperfeiçoaremos o exemplo para gerar e-mails em formato HTML. A idéia continua a mesma e as modificações serão mínimas.

> **Atenção:** *Será necessário que o leitor possua algum conhecimento prévio com a sintaxe da linguagem HTML, pois não entraremos em detalhes sobre as tags e códigos HTML. É necessário também o uso da biblioteca* ***Redemption Objects****, disponível em:* ***www.dimastr.com/redemption***

Criando uma página de modelo para o email

Vamos criar uma página HTML que servirá como base para o envio da mala direta. Você poderá utilizar qualquer editor de texto, como o Bloco de Notas, mas recomendo utilizar o Adobe Dreamweaver, por facilitar a criação. Crie o mesmo texto do artigo anterior e formate-o como desejar. Observe a **Figura 1.**

Mantemos as variáveis entre chaves, exatamente como no artigo anterior. Elas serão substituídas pelos dados da tabela de Funcionários do *Northwind*. Na **Listagem 1**, temos um código exemplo do modelo em formato HTML.

> **Atenção:** *Na mensagem modelo, uma imagem foi incluída na tag* ***<img src="outlook.jpg">****. Ao enviar uma mala direta, você deverá trocar o endereço da imagem para o endereço em que ela se encontra hospedada, por*

exemplo: ***<img src="http://www.dominio.com/email/imagens/outlook.jpg">****. Assim, quando o destinatário receber o e-mail, o Outlook realizará o download da imagem diretamente do servidor indicado.*

Alterando o projeto para enviar emails em HTML

No Delphi, abra o projeto do artigo anterior e configure a tela como a **Figura 2.**

Observe que retiramos o *Memo* que armazenava a mensagem em formato texto, e adicionamos um *Edit* ("edtArquivo") que armazenará o caminho completo do modelo. O usuário poderá selecionar o modelo que quiser através do *Selecionar Arquivo* ("btnAbrir").

Adicionaremos uma nova função na seção *private*, chamada *AbrirArquivo*, que será utilizada para abrir o arquivo de modelo e armazenar seu conteúdo em um *StringList*, para posterior substituição das variáveis e envio do e-mail.

Também criaremos uma nova variável, chamada *DirAplicativo*, para armazenar o diretório atual do aplicativo. Veja o código completo na **Listagem 2** (não esqueça de excluir o código onde consta o *Memo* removido anteriormente).

Listagem 1. Código HTML da página de modelo

```
<html>
<style>
  body{font-size:11px;font-family:Tahoma; margin:10px; }
  #mensagem{border:1px solid #999999;}
  #titulo{background:#F5D47C; padding:5px;
    font-weight:bold; border:1px solid white; }
  #imagem{float:left; margin-top:10px;
    margin-right:10px; margin-left:5px;}
  #corpo{padding:10px;}
  #agradecimento{padding-left:10px;}
  #rodape{padding-left:10px; padding-bottom:8px;}
  #linha{padding:0px}
  .titulo1{ font-size:12px; font-weight:bold;}
  .titulo2{ font-size:11px; font-style:italic}
</style>

<body>
 <div id="mensagem">
 <div id="titulo">Prezado [TRATAMENTO] [NOME] [SOBRENOME],
 </div>
 <div id="imagem"><img src="outlook.jpg"></div>
 <div id="corpo">
 <br>estamos lhe enviando esta mensagem com o intuito
     de divulgar nosso mais novo software para envio
     automático de emails e mala direta.
 <br><br><br>

 <p>Podemos destacar diversas funções, entre elas:</p>
 <li>Personalização de modelos;</li>
 <li>Envio em formato HTML;</li>
 <p>entre outros.</p>

 <p>Caso queira saber mais,
 <a href="mailto:email@dominio.com.br?subject=Quero saber mais">
  envie-nos um email</a>, que entraremos em contato assim que possível.
</p>
</div>

 <div id="agradecimento">
   Agradecemos sua atenção.
 </div>

 <div id="linha"><hr size="1" color="#999999"></div>
 <div id="rodape">
    <span class="titulo1">Sistemas</span>
    <br>
</div>
</div>

</body>

</html>
```

Apenas uma parte pequena do código foi alterada. Trocamos a propriedade *Body* para *HTMLBody* e a propriedade *BodyFormat* para *olFormatHTML* no código do *OnClick* do botão que envia o e-mail. Além disso, criamos uma função (*AbrirArquivo*) para ler o conteúdo do arquivo HTML e jogá-lo dentro de um *StringList*, para manipulação interna das variáveis e envio dos e-mails.

Também adicionamos um botão para permitir ao usuário escolher uma página HTML, e na variável *DirAplicativo* armazenamos o diretório atual do aplicativo. Execute a aplicação e faça o teste.

Anexando arquivos

Alteraremos o mesmo projeto para anexar um arquivo relacionado ao produto da mala direta (folder). Esse recurso poderá ser utilizado para empresas que desejam automatizar o processo de marketing de seus produtos e serviços.

Adicione um *Edit* ("edtAnexo"), um *Button* ("btnAnexar") e na propriedade *Caption*, digite "Inserir Anexo". Veja na **Figura 3** como ficou a aplicação.

Implemente o evento *OnClick* do botão de anexar, através da **Listagem 3**, que será responsável por criar uma janela para permitir ao usuário a escolha do anexo.

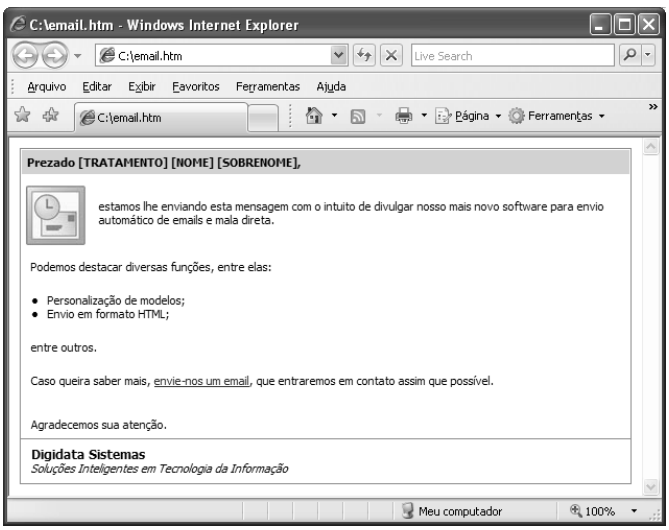


Figura 1. Visualizando o modelo da mensagem no navegador

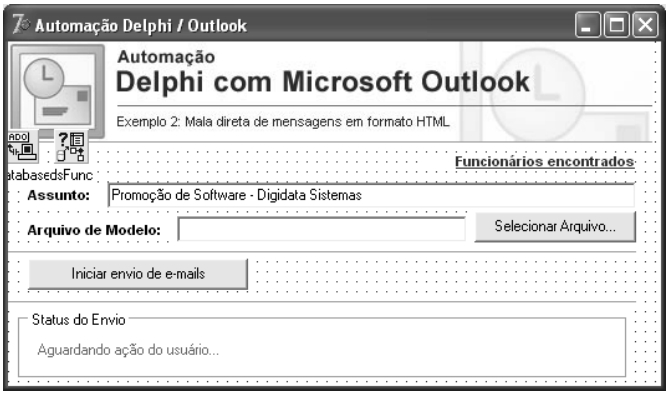


Figura 2. Tela principal do aplicativo modificado para utilizar modelos HTML

No evento *OnClick* do *Enviar*, após a linha *MI.HTMLBody := CorpoEmail;*, adicione o código a seguir, que será responsável por incluir o anexo na mensagem.

```
if FileExists(edtAnexo.Text) then
  oItem.Attachments.Add(edtAnexo.Text);
```

Agora rode o aplicativo, adicione um arquivo anexo e clique no botão de enviar os e-mails. Observe que para cada e-mail gerado, o Outlook anexou o arquivo escolhido no aplicativo (**Figura 4**).

Conclusão

Aperfeiçoamos o mesmo projeto com poucas alterações no código e adicionamos recursos para envio de mensagens em formato HTML e com anexos. No próximo artigo construiremos um banco de dados e um aplicativo para filtrar mensagens recebidas no Outlook e, através de filtros personalizados, armazenaremos as mensagens recebidas. Bons estudos e até a próxima. ■

Listagem 2. Código modificado do aplicativo

```
...
  private
    DirAplicacao: string;
    procedure AbrirArquivo(Arquivo: string;
      Lista: TStrings);
...
  public
  end;

var
  frmPrincipal: TfrmPrincipal;

Implementation

uses OleServer, OutlookXP, Redemption_TLB, ComObj;
{$R *.dfm}

procedure TfrmPrincipal.AbrirArquivo(Arquivo: string; Lista: TStrings);
var
  ArquivoTexto: TextFile;
  Conteudo: string;
begin
  Lista.Clear;
  AssignFile(ArquivoTexto, Arquivo);
  FileMode := fmOpenRead;
  Reset(ArquivoTexto);
  while not Eof(ArquivoTexto) do
  begin
    ReadLn(ArquivoTexto, Conteudo);
    Lista.Add(Conteudo);
  end;
  CloseFile(ArquivoTexto);
end;

...
procedure TfrmPrincipal.FormCreate(Sender: TObject);
begin
  dsFunc.Open;
  { Código novo }
  DirAplicacao := ExtractFileDir(Application.ExeName);
  edtArquivo.Text := DirAplicacao + '\email.htm';
end;

...
procedure TfrmPrincipal.btnAbrirClick(Sender: TObject);
var
  dlgAbrir: TOpenDialog;
begin
  dlgAbrir := TOpenDialog.Create(self);
  try
    dlgAbrir.InitialDir := DirAplicacao;
    dlgAbrir.Options := [ofFileMustExist];
    dlgAbrir.Filter  :=
      'Páginas da Web (htm,html)|*.htm;*.html';
    dlgAbrir.FilterIndex := 2;
    if dlgAbrir.Execute then
      edtArquivo.Text := dlgAbrir.FileName;
  finally
    dlgAbrir.Free;
  end;
end;

{ Código modificado }
procedure TfrmPrincipal.btnEnviarClick(Sender: TObject);
var
  Outlook: TOutlookApplication;
  NmSpace: NameSpace;
  MI, oItem, MAPIFolder, sFolder: OleVariant;
  TextoEmail: TStrings;
  sAssunto, sCampo, sDados: string;
  CorpoEmail: WideString;
  a, b, c: integer;
begin
  lblInfo.Caption := 'Conectando ao Outlook...';
  Application.ProcessMessages;
  try
    Outlook := TOutlookApplication.Create(nil);
    Outlook.ConnectKind := ckRunningOrNew;
    Outlook.Connect;
  except
    lblInfo.Caption :=
      'Erro ao conectar ao Outlook...';
    Application.ProcessMessages;
    Abort;
  end;

  try
    lblInfo.Caption := 'Buscando perfil de acesso...';
    Application.ProcessMessages;
    NmSpace := Outlook.GetNamespace('MAPI');
    NmSpace.Logon('', '', False, False);
    TextoEmail := TStringList.Create;
    sAssunto := edtAssunto.Text;
    { Para cada registro, enviar um e-mail }
    lblInfo.Caption := 'Percorrendo registro de funcionários...';
    Application.ProcessMessages;
    while not dsFunc.EOF do
    begin
      lblInfo.Caption := 'Gerando mensagem para: ' +
        dsFunc.FieldByName('NOME').AsString;
      Application.ProcessMessages;
      { Limpando as variáveis }
      CorpoEmail := '';
      TextoEmail.Clear;
      AbrirArquivo(edtArquivo.Text, TextoEmail);
      { TextoEmail.Assign(memTexto.Lines); }
      { Localizando e substituindo variáveis entre chaves [] por dados }
      for a:= 0 to TextoEmail.Count-1 do
       for b:= 0 to dsFunc.Fields.Count-1 do
       begin
         sCampo := Uppercase(dsFunc.Fields[b].FieldName);
         sDados := dsFunc.FieldByName(sCampo).AsString;
         if pos('['+ sCampo +']',TextoEmail[a]) > 0
         then
           TextoEmail[a] := SubstituirCampoTexto(
             TextoEmail[a], '['+ sCampo +']', sDados);
      end;
      oItem := Outlook.CreateItem(0);
      MI := CreateOleObject('Redemption.SafeMailItem');
      MI.Item := oItem;
      { O formato agora é HTML }
      MI.Item.BodyFormat := olFormatHTML;
      MI.Recipients.Add(dsFunc.FieldByName('EMAIL').AsString);
      MI.Subject := sAssunto;
      for c:= 0 to TextoEmail.Count-1 do
        CorpoEmail := CorpoEmail + #13 + TextoEmail.Strings[c];
        { Alteramos aqui para HTMLBody }
        MI.HTMLBody := CorpoEmail;
        MI.Send;
        dsFunc.Next;
    end;
    lblInfo.Caption :=
      'Concluindo criação de mensagens...';
    Application.ProcessMessages;
    MAPIFolder := Outlook.Session.GetDefaultFolder(olFolderInbox);
    sFolder := CreateOleObject(
      'Redemption.MAPIFolder');
    sFolder.Item := MAPIFolder;
    sFolder.Display;
  finally
   lblInfo.Caption := 'Limpando variáveis...';
   Application.ProcessMessages;
   try
    nmSpace.LogOff;
   except
   end;

   try Outlook.Disconnect;
   except
   end;
  end;

  lblInfo.Caption := 'Operação concluída com sucesso...';
  Application.ProcessMessages;
end;
end.
```

Listagem 3. Evento OnClick do Inserir Anexo

```
var
 dlgAnexo: TOpenDialog;
begin
  dlgAnexo := TOpenDialog.Create(self);
  try
    dlgAnexo.InitialDir := DirAplicacao;
    dlgAnexo.Options := [ofFileMustExist];
    dlgAnexo.Filter :=
      'Arquivos de Folder (pdf,doc,xls)|*.pdf; '+
      '*.doc;*.xls';
    dlgAnexo.FilterIndex := 2;
    if dlgAnexo.Execute then
      edtAnexo.Text := dlgAnexo.FileName;
  finally
    dlgAnexo.Free;
  end;
end;
```

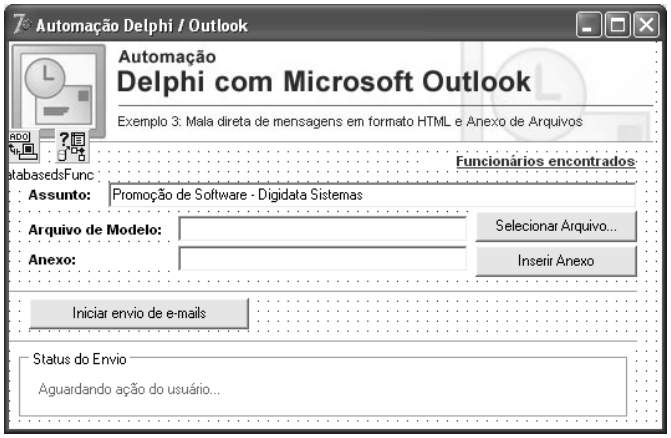


Figura 3. Interface principal após as alterações efetuadas

Links

Dicas de automação em Delphi
delphi.about.com/od/kbcontrolole/OLE_Automation_using_Delphi.htm
djpate.freeserve.co.uk/Automation.htm

Dicas de automação Outlook via Delphi
www.djpate.freeserve.co.uk/AutoOutl.htm

Dúvidas
E-mail: murilo.filho@asa-sistemas.com
Web site: www.asa-sistemas.com

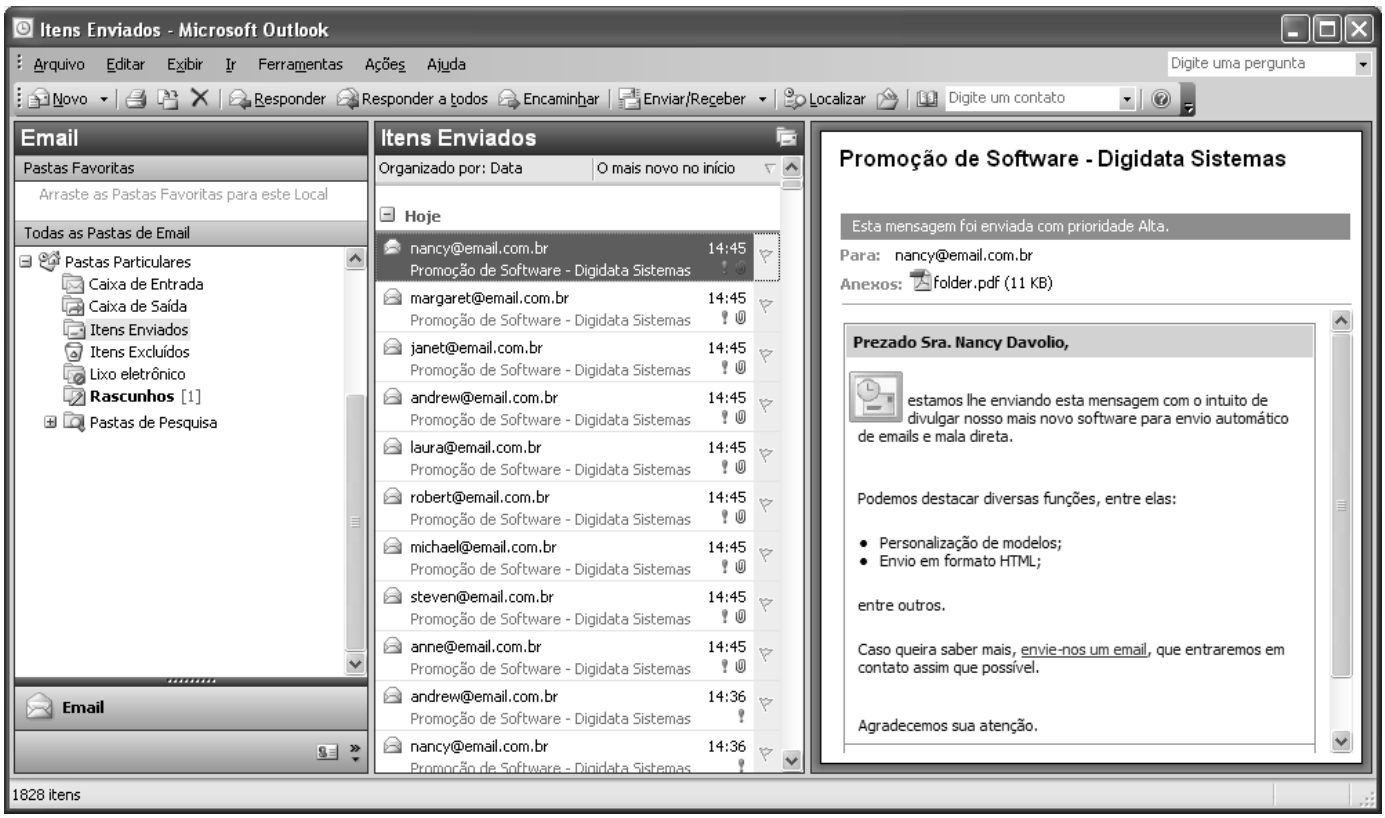
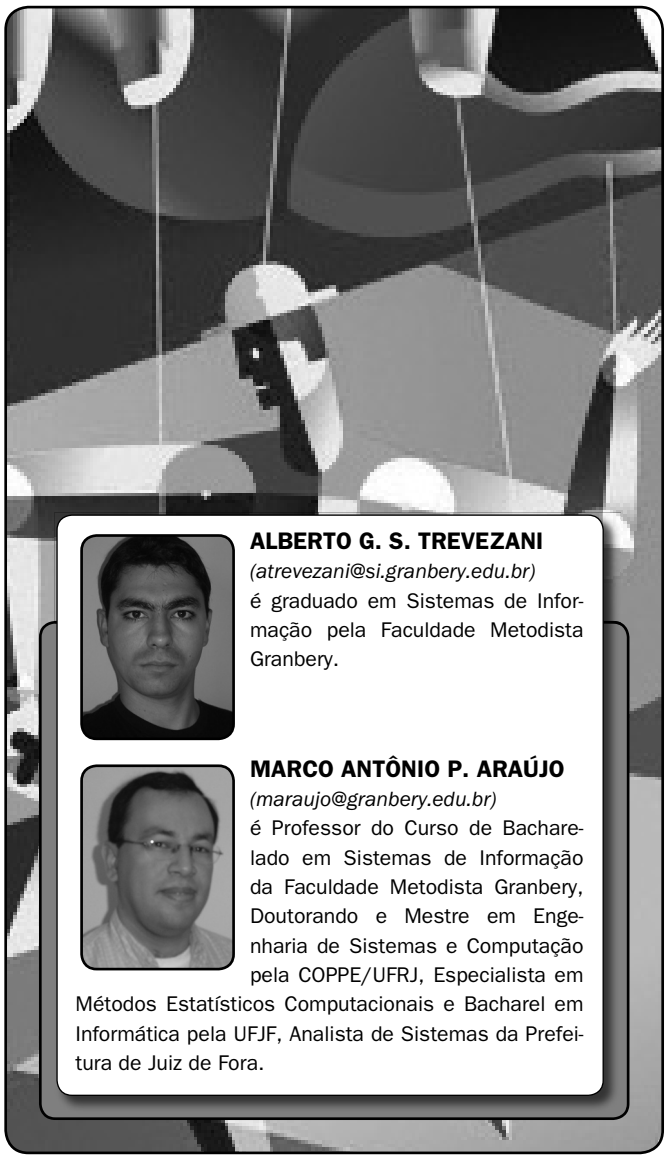


Figura 4. Mensagens enviadas no Outlook (observe o anexo)

Modelagem UML com Together



ALBERTO G. S. TREVEZANI
(atrevezani@si.granbery.edu.br)
é graduado em Sistemas de Informação pela Faculdade Metodista Granbery.

MARCO ANTÔNIO P. ARAÚJO
(maraujo@granbery.edu.br)
é Professor do Curso de Bacharelado em Sistemas de Informação da Faculdade Metodista Granbery, Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ, Especialista em Métodos Estatísticos Computacionais e Bacharel em Informática pela UFJF, Analista de Sistemas da Prefeitura de Juiz de Fora.

Com a difusão do desenvolvimento Orientado a Objetos e a necessidade de produzir software com mais qualidade, os diagramas UML (*Unified Modeling Language*) estão sendo cada vez mais utilizados no desenvolvimento de software, não apenas com o objetivo de documentação, mas também de planejar e gerenciar o desenvolvimento de aplicações.

A UML é uma evolução das representações tradicionais para análise e projeto de sistemas Orientados a Objetos, unificando as metodologias de representação de Booch, Rumbaugh e Jacobson, sendo adotada como padrão pela OMG (*Object Management Group*).

Para facilitar sua utilização, alguns IDEs, como o Delphi, incorporaram facilidades para a construção de seus diagramas e oferecem mecanismos para integrá-los ao processo de desenvolvimento.

Nesse sentido, este artigo apresenta as principais características de modelagem UML da Together, ferramenta integrada ao Delphi 2006, sendo proposta a modelagem parcial de um Sistema de Controle Acadêmico, onde será demonstrada a elaboração do diagrama de casos de uso, de classes e de seqüência.

Requisitos do Sistema de Controle Acadêmico

Para esse estudo de caso, será considerado um fragmento de um Sistema de Controle Acadêmico, considerando os seguintes requisitos:

- O sistema deve permitir à Secretaria o cadastramento de cursos, contendo código e descrição;
- O sistema deve permitir à Secretaria o cadastramento de disciplinas de cursos, contendo ainda código e descrição;
- O sistema deve permitir à Secretaria o cadastramento de professores, contendo código, nome e CPF;
- O sistema deve permitir à Secretaria o cadastramento de aluno, contendo código, nome e matrícula;

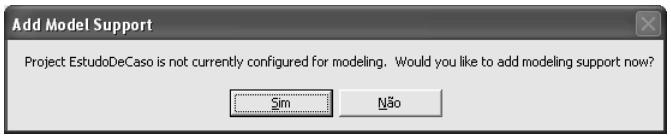


Figura 1. Confirmação para ativação do suporte a modelagem

• O sistema deve permitir à Secretaria matricular alunos em disciplinas de cursos, contendo ainda o ano e o semestre.

Esses requisitos são a base para a construção dos diagramas de casos de uso e de classes do estudo de caso. O requisito referente à matrícula de alunos será explorado na construção do diagrama de seqüência.

Diagrama de Casos de Uso

O Diagrama de Casos de Uso tem por objetivo principal mostrar as principais funcionalidades de um sistema, bem como evidenciar os papéis responsáveis por essas funcionalidades. Assim, uma funcionalidade é chamada de um Caso de Uso, enquanto um papel é representado por um ator.

Dessa forma, definidos os requisitos da aplicação, deve-se iniciar um novo projeto no Delphi. Para isso basta acessar a opção de menu *File>New>VCL Forms Application - Delphi for Win32*. Dê o nome de "EstudoDeCaso.dpr" e a unit como "untPrincipal.pas", que conterá a interface da aplicação.

Como o suporte a modelagem não é habilitado na criação do projeto, essa ativação deverá ser feita manualmente clicando na aba *Model View*. Será perguntado se deve ser adicionado ao projeto o suporte a modelagem, bastando responder que Sim (**Figura 1**).

Feito isso, o Delphi acionará a aba *Model View* (**Figura 2**).

Ao ativar o suporte à modelagem, o Delphi adicionará automaticamente ao *Model View* os namespaces já existentes no projeto atual (*EstudoDeCaso*). Nesse processo é criado ainda um namespace para cada unit do projeto, como o caso da *untPrincipal*.

Antes da criação do primeiro diagrama, o de casos de uso, será criado um namespace para o diagrama. Para isso, basta selecionar o projeto no *Model View,* pressionar o botão direito e escolher a opção *Add>Namespace*. Feito isso, deve-se alterar o nome padrão para "Academico".

Com o namespace criado, já é possível construir o diagrama de casos de uso da aplicação bastando selecionar o namespace criado, pressionar o botão direito e escolher a opção *Add>Other Diagram*.

Na tela apresentada, basta selecionar *Use Case Diagram* e definir um nome para o mesmo (**Figura 3**).

Ao selecionar um diagrama de casos de uso, o Together disponibiliza uma aba de ferramentas (*Tool Palette*) relacionadas com o diagrama em questão (**Figura 4**).

Deverá ser arrastado para o diagrama de caso de uso um *Actor* com o nome de "Secretaria" e cinco *Use Case,* descrevendo-os como: "Cadastrar Professor", "Cadastrar Curso", "Cadastrar Disciplina", "Cadastrar Aluno" e "Matricular Aluno".

Para ligar um ator a um caso de uso basta selecionar o *Communicates*, em seguida selecionar o ator *Secretaria* e o caso de

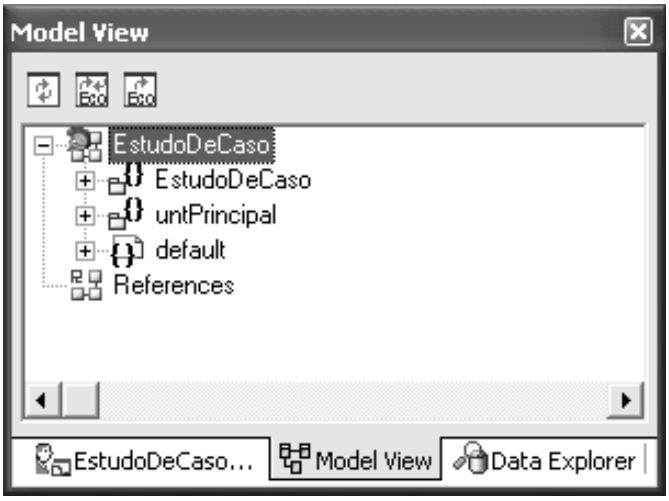


Figura 2. Model View habilitado

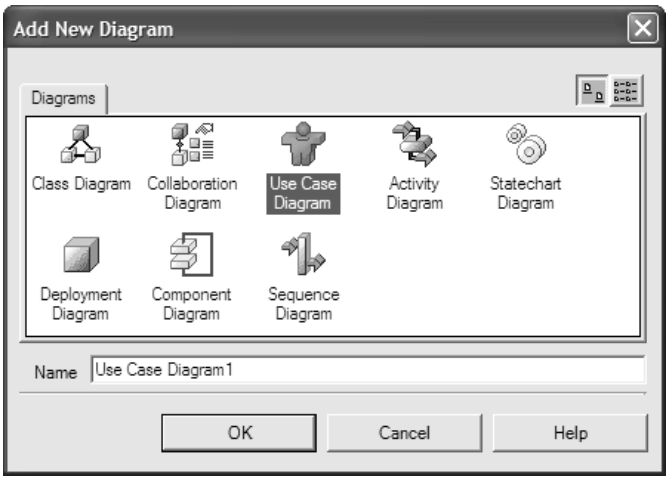


Figura 3. Definindo o diagrama de Casos de Uso

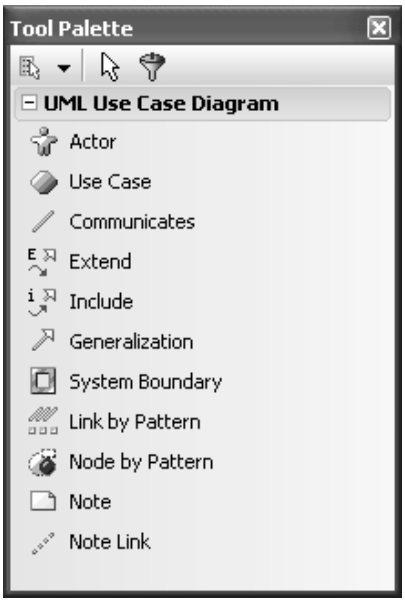


Figura 4. Componentes para o diagrama de Casos de Uso

uso a ser relacionado. Deve-se então repetir esse procedimento para as demais ligações.

Para que a seta apareça, representando o sentido da ligação, basta selecionar a opção *Directed*, presente nas propriedades do *Object Inspector*, definindo-a como *True*. O resultado final desse diagrama é apresentado na **Figura 5**.

Diagrama de Classes

O Diagrama de Classes representa os objetos do domínio da aplicação, numa visão Orientada a Objetos. Nesse diagrama, objetos com mesmas características (atributos e métodos) são agrupados em classes, que podem se relacionar com outras classes através de herança ou associação, por exemplo.

O próximo passo refere-se à criação do diagrama de classes e, para isso, basta selecionar o namespace *Academico*, pressionar o botão direito e escolher a opção *Add>Class Diagram*, adicionando ao projeto um diagrama de classes.

Ao selecionar um diagrama desse tipo, a *Tool Palette* será semelhante a **Figura 6**.

Seguindo o cenário proposto, serão adicionadas ao diagrama de classes as classes TPessoa, TAluno, TProfessor, TCurso, TDisciplina e TMatricula. Para adicionar ao diagrama essas classes, basta selecionar o *Class* na *Tool Palette* e arrastar para o diagrama.

Deverão ser adicionadas às classes, os atributos relacionados na **Tabela 1**. Para adicionar um atributo, basta selecionar a classe, pressionar o botão direito e escolher a opção *Add>Field*.

Classe	Atributos
Tpessoa	Codigo: integer Nome: string
Taluno	Matricula: string
Tprofessor	CPF: string
Tcurso	Codigo: integer Descricao:string
Tdisciplina	Codigo: integer Descricao: string
Tmatricula	Ano: integer Semestre: integer Disciplinas: TList

Tabela 1. Atributos das classes

Com os atributos definidos já é possível definir os relacionamentos entre as classes. As classes *TAluno* e *TProfessor* possuem características comuns e herdam da classe *TPessoa*. Para criar a herança entre elas, basta selecionar no *Tool Palette* o *Generalization*, selecionar a classe *TAluno* e em seguida a classe *TPessoa*. Com isso a herança será criada, devendo-se repetir o mesmo procedimento para a classe *TProfessor*.

A ordem de seleção das classes no diagrama para a construção de heranças e associações influencia no diagrama, pois é essa ordem que definirá o sentido do relacionamento. A definição de uma associação é idêntica, à definição de uma herança, o que difere é que o componente para essa finalidade é o *Association*.

Para definir uma associação de multiplicidade 1:1 entre classes, basta selecionar o *Association* e em seguida selecionar as classes origem e destino. Feito isso, será criada na classe origem um atributo de navegação com um nome default, devendo esse nome ser modificado.

Esse atributo de navegação será do mesmo tipo que a classe relacionada, tendo como função prover a ligação através de um ponteiro para um objeto. As multiplicidades podem ser definidas selecionando o relacionamento criado e alterando no *Object Inspector* as propriedades *Client Cardinality* (cardinalidade da classe origem) e *Supplier Cardinality* (cardinalidade da classe destino).

Para restringir a navegabilidade entre as classes, basta alterar para *True* a propriedade *Directed* no *Object Inspector*. As associações devem ser feitas conforme **Tabela 2**.

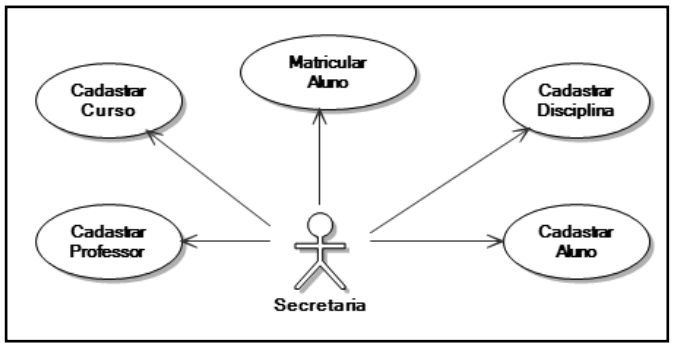


Figura 5. Resultado final do diagrama de Caso de Uso

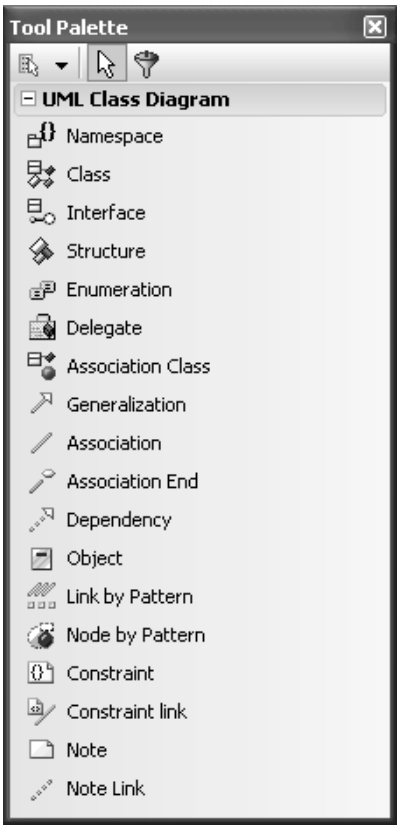


Figura 6. Componentes para o diagrama de classes

Classe origem	Classe destino	Multiplicidade Origem	Multiplicidade Destino	Nome Relacionamento
TDisciplina	TProfessor	0..1	1	Professor
TDisciplina	TCurso	0..*	1	Curso
TMatricula	TAluno	0..*	1	Aluno
TMatricula	TCurso	0..*	1	Curso

Tabela 2. Associações entre as classes

Nome do Caso de Uso	Matricular Aluno		
Descrição	Esse caso de uso permite a matrícula de alunos em disciplinas de um curso.		
Ator Envolvido	Secretaria		
Interação entre Ator e Sistema	Secretaria		Sistema
	O caso de uso é iniciado quando a Secretaria seleciona a opção Matricular Aluno.		
			Sistema apresenta janela com os campos aluno (RI01), curso (RI02), disciplinas (RI03), ano e semestre.
	Secretaria preenche os campos e seleciona a opção Efetuar Matrícula.		
			Sistema valida as informações preenchidas pela secretária (EX01) (RN01).
			Sistema matricula aluno (EX02) e volta para a tela inicial. Nesse momento, esse caso de uso é encerrado.
Exceções	EX01	Os campos devem estar todos preenchidos. Se houver problemas no preenchimento do formulário o sistema exibirá uma mensagem de erro adequada.	
	EX02	Caso o aluno já se encontre matriculado em alguma das disciplinas selecionadas, o sistema deverá apresentar uma mensagem apropriada.	
Alternativas	AL01	A Secretaria pode cancelar a operação de matrícula a qualquer momento, fechando a janela.	
Regras de Negócio	RN01	Um aluno não pode se matricular em mais do que cinco disciplinas num mesmo ano e semestre.	
Requisitos de Interface com o Usuário	RI01	O sistema deve apresentar os alunos cadastrados numa caixa de combinação, para serem escolhidos pela Secretaria.	
	RI02	O sistema deve apresentar os cursos cadastrados numa caixa de combinação, para serem escolhidos pela Secretaria.	
	RI03	O sistema deve apresentar as disciplinas cadastradas, do curso selecionado, em uma caixa de listagem, para serem escolhidas pela Secretaria.	

Tabela 3. Descrição do caso de uso Matricular Aluno

Os nomes escolhidos para representar os relacionamentos indicam o nome da classe com que se relacionam. Por exemplo, a classe *TDisciplina* se relacionará com a classe *TProfessor* e, para facilitar o entendimento, foi utilizado o nome "Professor" para o atributo de navegação.

O próximo passo é definir a associação de multiplicidade 1:N entre as classes *TMatricula* e *TDisciplina*. Para isso, o atributo *Disciplinas* será o atributo de ligação, sendo uma lista que armazenará vários objetos.

Para definir uma associação 1:N basta selecionar o atributo *Disciplinas*, definido na classe *TMatricula*, selecionar o atributo *Associates* no *Object Inspector* e localizar a classe *TDisciplina* (*Academico.TDisciplina*). Feito isso basta alterar a multiplicidade de origem para "0..1" e a de Destino para "1..*".

As associações no Delphi por padrão são unidirecionais, isso é, a visibilidade é apenas em um sentido. Para que as duas classes envolvidas na associação tenham acesso entre si é preciso criar dois relacionamentos, onde cada um dos relacionamentos deverá ter uma das classes como origem.

Os métodos serão definidos na construção do diagrama de seqüência, facilitando encontrá-los. O resultado final do diagrama de classes é representado na **Figura 7**.

Diagrama de Seqüência

A etapa seguinte refere-se à construção do diagrama de seqüência, que tem o objetivo de detalhar o comportamento de um determinado caso de uso, numa visão Orientada a Objetos. Assim, nesse estudo de caso, será construído o diagrama de seqüência para o caso de uso *Matricular Aluno*, conforme especificado na **Tabela 3**.

O formato como a especificação de casos de uso foi apresentada nessa tabela é uma sugestão. Existem vários *templates* diferentes que podem ser utilizados para esse fim. De qualquer forma, o importante é descrever os requisitos do sistema com um maior grau de detalhamento, permitindo um melhor entendimento do problema e oferecendo melhores mecanismos para o planejamento do desenvolvimento.

Nesse formato utilizado, vale a pena observar:

• *Nome do caso de uso*: deve conter o mesmo nome do caso de uso que está no diagrama de casos de uso e que será detalhado;

• *Descrição*: contém um resumo da utilidade do caso de uso;

• *Ator envolvido*: é o ator quem executa o caso de uso conforme representado no diagrama de casos de uso. Em determinadas situações, pode haver mais de um ator envolvido;

• *Interação entre o ator e o sistema*: descreve os passos envolvidos na realização do caso de uso, evidenciando as responsabilidades do ator e do sistema, num processo interativo;

• *Exceções*: indicam situações onde primordialmente o tratamento de erros deve ser efetuado;

• *Alternativas*: indicam situações opcionais que podem ocorrer durante a utilização do caso de uso;

• *Regras de Negócio*: são as regras impostas para a utilização do caso de uso, definidas pelo domínio da aplicação;

• *Requisitos de Interface com o Usuário*: descrevem características que devem ser implementadas na interface com o usuário.

Partindo desse detalhamento, pode-se construir mais facilmente o diagrama de seqüência. Para isso, basta selecionar o namespace *Academico*, pressionar o botão direito e escolher a opção *Add>Other Diagram*.

Na janela apresentada, seleciona-se o diagrama *Sequence Diagram*, definindo um nome para o mesmo, neste exemplo "Matricular Aluno" (**Figura 8**).

Ao selecionar um diagrama de seqüência, a *Tool Palette* se adequará ao diagrama (**Figura 9**).

Deve-se então adicionar ao diagrama de seqüência o ator. Para isso, basta selecionar *Actor* e arrastá-lo para o diagrama, alterando seu nome para "Secretaria". O primeiro objeto a ser criado deverá ser o formulário de matrícula, representando a interface da aplicação.

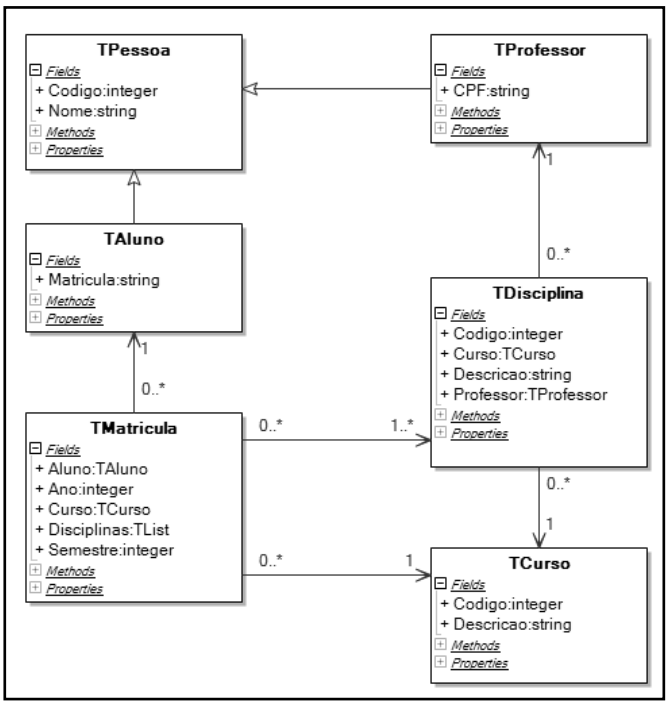


Figura 7. Resultado final do Diagrama de Classes

Selecione na *Tool Palette* o *Object*, arrastando-o para o diagrama, alterando seu nome para "frmMatricularAluno". Para criar uma ligação entre o ator e o objeto basta selecionar o *Message* e em seguida selecionar o ator e posteriormente o objeto.

Como o formulário será criado pelo Ator, deve-se selecionar a ligação e alterar no *Object Inspector* a propriedade *Label* para "Exibir". Percebe-se que esse objeto não faz parte do domínio da aplicação e, por isso, não está presente no diagrama de classes. Por isso, o nome da ligação aparece prefixado com "//".

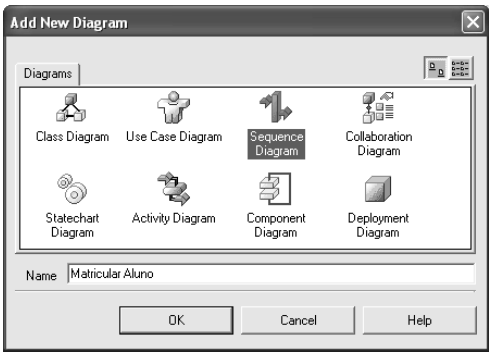


Figura 8. Definindo o Diagrama de Seqüência

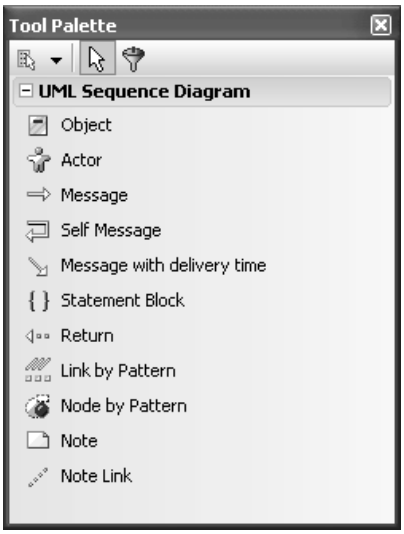


Figura 9. Componentes para o diagrama de seqüência

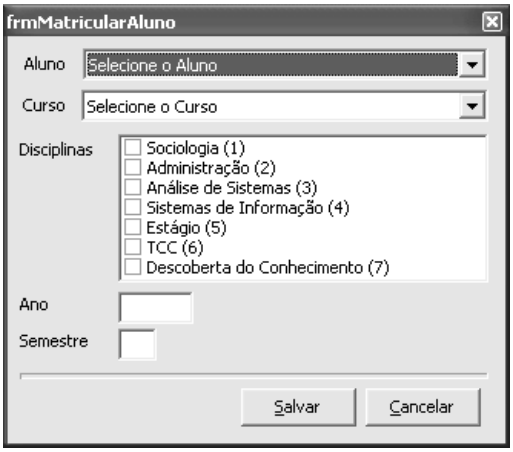


Figura 10. Janela de matrícula de aluno

O formulário será a tela de Matrícula de Alunos, onde será apresentada uma caixa de combinação contendo os alunos cadastrados, outra contendo os cursos, e uma lista das disciplinas disponíveis, além de campos para a entrada do ano e semestre da matrícula. Apenas com o intuito de facilitar essa visualização, a **Figura 10** exibe um protótipo desse formulário.

Retornando ao diagrama, adiciona-se então um conjunto de mensagens entre o ator e o formulário. Esses métodos serão responsáveis pela entrada dos dados por parte do ator. Adiciona-se agora mais um objeto ao diagrama e, através da propriedade *Instantiates* no *Objet Inspector*, deve-se localizar a classe *TAluno*.

Seu conteúdo deverá ser apenas *Academico.TAluno*, limpando o seu nome, representando assim a classe ao invés de um objeto, uma vez que, nesse processo, o formulário solicitará à classe uma lista contendo todos os alunos. Para representar esse processo deve-se selecionar na *Tool Palette* o *Message* e em seguida selecionar os objetos *frmMatriculaAluno* e *TAluno*.

Para criar esse método basta selecionar a propriedade *Operation* no *Object Inspector* e definir a assinatura como "GetAlunos():TList". Para que seja criado o método na classe, basta selecionar o botão *Create* para a mensagem que será apresentada (**Figura 11**). Não é possível criar um método usando esse procedimento já com parâmetros, que deverão ser adicionados manualmente.

Adiciona-se a seguir uma ligação entre o ator e o formulário, alterando a propriedade *Label* para "SelecionarAluno". Esse processo representa a seleção de um aluno através do formulário. O próximo passo será adicionar ao diagrama de seqüência três objetos alterando respectivamente a propriedade *Instantiates* para *TCurso*, *TDisciplina* e *TMatricula*.

Nos objetos *TCurso* e *TDisciplina* o conteúdo da propriedade *Name* deverá estar em branco e no *TMatricula* a propriedade *Name* deverá ser definida como "novaMatricula". As demais ligações a serem construídas entre os objetos estão representadas na **Tabela 4**. Percebe-se que as operações criadas a partir da ferramenta são funções, com um valor de retorno.

Classe	Método	Params
TMatricula	SetAluno()	pAluno: TAluno
TMatricula	SetCurso()	pCurso: TCurso
TMatricula	SetDisciplinas()	pDisciplinas: TList
TMatricula	SetAno()	pAno: Integer
TMatricula	SetSemestre()	pSemestre: Integer

Tabela 5. Descrição dos parâmetros

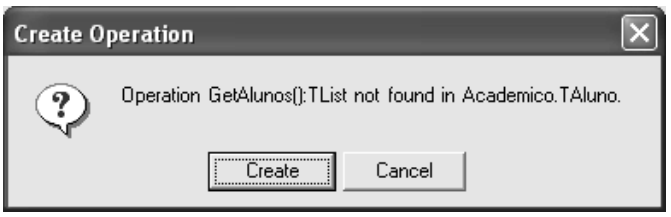


Figura 11. Criação de um método

Listagem 1. Código fonte gerado.

```
unit Academico;
interface
{ Declare no uses as units }
uses
  Classes, SysUtils;
type
  TProfessor = class;
  TDisciplina = class;
  TCurso = class;
  TPessoa = class
  public var
    Codigo: integer;
    Nome: string;
  end;
  TAluno = class(TPessoa)
  public
    class function GetAlunos: TList;
  var
    Matricula: string;
  end;
  TCurso = class
  public
    class function GetCursos: TList;
    class function GetDisciplinas: TList;
  var
    Codigo: integer;
    Descricao: string;
  end;
  TMatricula = class
  public
    class function ChecaJaMatriculado: Boolean;
    function SetAluno(pAluno: TAluno): Boolean;
    function SetCurso(pCurso: TCurso): Boolean;
    function SetDisciplinas(pDisciplinas: TList): Boolean;
    function SetSemestre(pSemestre: integer): Boolean;
    function SetAno(pAno: integer): Boolean;
    function persistir: Boolean;
  var
    Aluno: TAluno;
    Ano: integer;
    Semestre: integer;
    Disciplinas: TList;
    Curso: TCurso;
  end;
  TDisciplina = class
  public
    class function GetDisciplinas: TList;
  var
    Codigo: integer;
    Descricao: string;
    Professor: TProfessor;
    Curso: TCurso;
  end;
  TProfessor = class(TPessoa)
  public var
    CPF: string;
  end;
implementation
class function TMatricula.ChecaJaMatriculado: Boolean;
begin
end;
function TMatricula.SetAluno(pAluno: TAluno): Boolean;
begin
end;
function TMatricula.SetCurso(pCurso: TCurso): Boolean;
begin
end;
function TMatricula.SetDisciplinas(pDisciplinas: TList):Boolean;
begin
end;
function TMatricula.SetSemestre(pSemestre: integer): Boolean;
begin
end;
function TMatricula.SetAno(pAno: integer): Boolean;
begin
end;
class function TAluno.GetAlunos: TList;
begin
end;
class function TCurso.GetCursos: TList;
begin
end;
class function TDisciplina.GetDisciplinas: TList;
begin
end;
class function TCurso.GetDisciplinas: TList;
begin
end;
function TMatricula.persistir: Boolean;
begin
end;
end.
```

No diagrama de classes deverá ser alterada a propriedade *Params* de alguns métodos, conforme a **Tabela 5**.

Os métodos *GetAlunos*, *GetCursos* e *GetDisciplinas*, respectivamente das classes Aluno, Curso e Disciplina, deverão ser métodos de classe, pois retornarão para o formulário uma lista de objetos da classe requerida. Para mudar um método normal para método de classe basta selecionar o método desejado e alterar a propriedade *Class* para *True*.

O *ChecaJaMatriculado* será responsável por verificar se o Aluno já se encontra matriculado. Ele também será um método de classe. Voltando ao diagrama de seqüência, o objeto *TMatricula*, responsável por matricular o aluno, será instanciado pelo formulário.

Para representar essa condição, deve-se mudar a propriedade *Creation* da ligação para *True*. O resultado final do diagrama de seqüência é representado na **Figura 12**.

A **Listagem 1** contém o código gerado após a construção dos diagramas. Os métodos devem então ser implementados de forma apropriada.

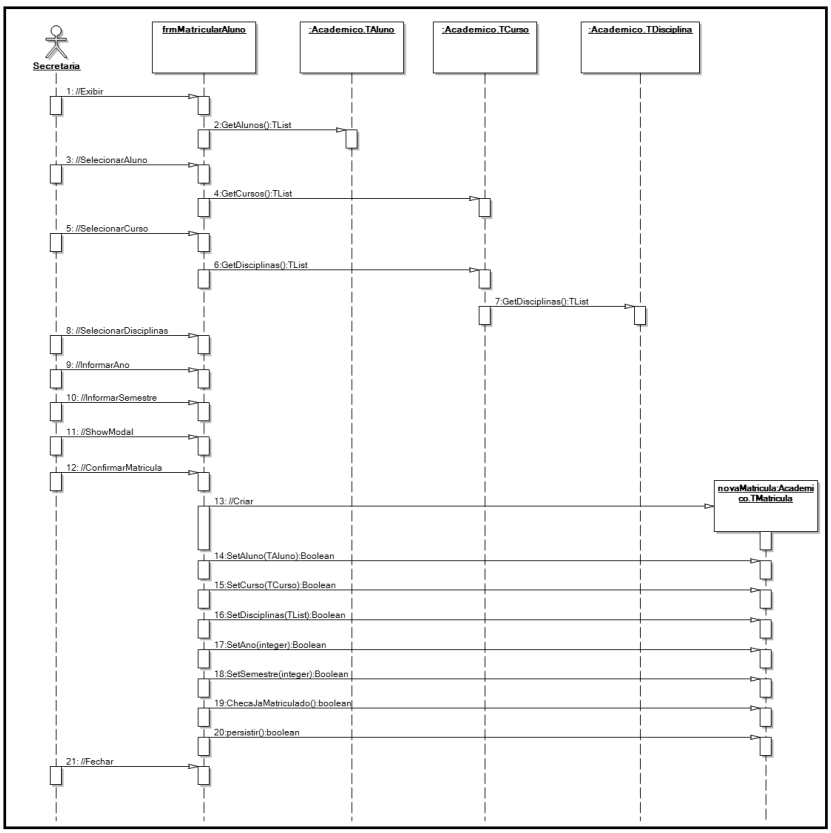


Figura 12. Resultado final do Diagrama de Seqüência

Conclusão

O objetivo deste artigo foi mostrar algumas características da ferramenta Together na modelagem de sistemas Orientados a Objetos através da construção de diagramas UML, de forma integrada no Delphi 2006.

Para maior entendimento desse processo, foi desenvolvido um estudo de caso onde se procurou apresentar de maneira prática os principais recursos dessa ferramenta. ■

Links

Site do Together

www.borland.com/br/products/together

Origem	Destino	Label	Operation
frmMatricularAluno	TCurso		GetCursos():TList
Secretaria	frmMatricularAluno	SelecionarCurso	
frmMatricularAluno	TCurso		GetDisciplinas():TList
TCurso	TDisciplina		GetDisciplinas():TList
Secretaria	frmMatricularAluno	SelecionarDisciplinas	
Secretaria	frmMatricularAluno	InformarAno	
Secretaria	frmMatricularAluno	InformarSemestre	
Secretaria	frmMatricularAluno	ShowModal	
Secretaria	frmMatricularAluno	ConfirmarMatricula	
frmMatricularAluno	TMatricula	Criar	
frmMatricularAluno	TMatricula		SetAluno():Boolean
frmMatricularAluno	TMatricula		SetCurso():Boolean
frmMatricularAluno	TMatricula		SetDisciplinas():Boolean
frmMatricularAluno	TMatricula		SetAno():Boolean
frmMatricularAluno	TMatricula		SetSemestre ():Boolean
frmMatricularAluno	TMatricula		ChecaJaMatriculado():Boolean
frmMatricularAluno	TMatricula		persistir():Boolean
Secretaria	frmMatricularAluno	Fechar	

Tabela 4. Trocas de mensagens entre os objetos do diagrama de seqüência

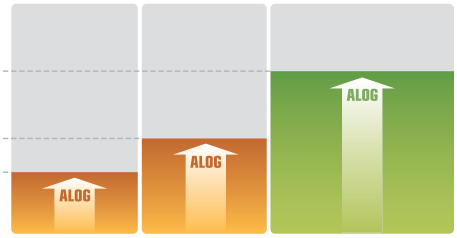
FAÇA COMO A ALOG
FOQUE EM SEU NEGÓCIO

A ALOG é totalmente focada em serviços de Data Center. Dedica-se 24 horas por dia, 7 dias por semana a cuidar do negócio dos nossos 320 clientes corporativos.

ALOG desenvolveu no Brasil o conceito de **HOSTING GERENCIADO**: um serviço de excelência que garantirá a sua empresa estabilidade, agilidade, redundância, e escalabilidade. Para que isso seja verdade os clientes da ALOG contam com **Suporte Incondicional***, um conceito revolucionário em atendimento no Brasil.

ENTENDA O **HOSTING GERENCIADO**

CAMADAS DE NEGÓCIO	DESCRIÇÃO
Processo de Negócio	Atividades cotidianas do cliente
Aplicação do Cliente	Softwares de Gestão (Ex: ERP, CRM, outros)
Ambiente da Aplicação	.NET, JAVA, SQL Server, MySQL, Oracle, Exchange
Sistema Operacional	Windows, Linux, Unix, Solaris
Servidor ou equipamento	Servidores RISC/CISC, Appliances, Roteadores, Switches
Rede	IP, Wan, Rede Local
Infra-estrutura Básica	Energia, HVAC, Segurança - 2N+1



ESTRATÉGIA DO CLIENTE	COLOCATION	HOSTING	HOSTING GERENCIADO

PLANOS DE **HOSTING GERENCIADO**

GERENCIADO 1

- **Pentium D 2.8 GHz**
- 512 RAM DDR2
- 2 HD SATA 80 GB
- 450 GB de Transf. Mensal
- Firewall e Backup

BOOT REMOTO
GERENCIAMENTO**

R$1.050,00 mensais
INSTALAÇÃO IMEDIATA***

GERENCIADO 2

- **Dual Xeon Dual Core 3.0 GHz**
- 1GB RAM DDR2 533
- 2 HD SATA RAID I 300 GB
- 1.000 GB de Transf. Mensal
- Firewall e Backup

BOOT REMOTO
GERENCIAMENTO**

R$2.850,00 mensais
INSTALAÇÃO IMEDIATA***

* Sujeito a condições contratuais ** Podendo escolher: HD, RD e GT *** Sujeito a disponibilidade no estoque

ALOG data centers

Excelência em Hosting
www.alog.com.br :: 0800 282 3330

Hosting Gerenciado | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais

Quantas cópias de seu software existem no mercado?

Esta é a chave mais segura do mundo contra pirataria de software.

Comprove você mesmo!

SafeNet®
The Foundation of Information Security

Alameda Tocantins, 280 - Barueri-SP

Tel: +55 11 4208-7700

http://br.safenet-inc.com - Contato: infobrasil@safenet-inc.com